Secure Multiparty Computation : Classical vs. Quantum

Asher Toback Henry Cooney

CS510 : Intro to Quantum Computing

Introduction

Secure multiparty computing (SMC) is a subfield of cryptography which allows multiple parties to compute a function, requiring shared inputs, without learning the secret input of any other participant. An example would be secure, distributed voting systems : using SMC, each voter can independently verify the results of an election, without learning how any other party voted.

Classical SMC has been the subject of considerable research effort in recent years. However, with the arrival of quantum computers on the horizon, it's natural to wonder how the SMC landscape might change. Early results show evidence that the use of a quantum communication infrastructure might offer substantial advantages in cryptography and secure communication. For example, *position based cryptography* (PBC) seeks to authenticate a user by their spatial location. Achieving this in a classical setting is difficult, as it has been proven that a large enough coalition of adversaries can 'spoof' a classical PBC protocol (18). However, a quantum PBC protocol may offer greater security (possibly requiring exponentially more EPR pairs to attack) (14). It is therefore natural to examine the relationship between quantum computing and SMC. Several important questions arise : is SMC possible in a quantum setting ? What advantages can quantum SMC offer over classical, if any ? Finally, how secure are SMC constructions (classical or quantum) in the face of a quantum attacker ?

In this paper, we focus on the security of Classical SMC protocols against quantum attacks. We start by outlining the building blocks of Classical SMC and explore some important results in this setting. We finish by examining what new constraints we must add to our model to ensure these protocols continue to be secure against quantum adversaries.

Classical SMC

The purpose of SMC protocols is to replace a trusted third party. That is, we hope to devise protocols which will allow us to cooperatively compute functions which would traditionally require us to use an external party to evaluate securely. In this way, SMC aims to achieve two goals ; namely, that at the end of a protocol the correct result is output, and that privacy is maintained throughout the protocol. The correctness of a protocol is simple to confirm, we only need to verify that the output is what we expect it to be. Privacy on the other hand is more difficult to ensure. What exactly do we mean by privacy, and how exactly can we check that it's been maintained ? Before we formalize this concept, let's first see an example of an SMC protocol at work.

1 A Simple SMC Protocol

Let Asher, Henry, and Fang each have a number that they would like to keep secret, but which they like to use in a computation with each other. In this case, let's assume they want to find the total sum of their three secret numbers. They could certainly achieve this if they sent them to a trusted party who would perform the addition and output the result to them all. However, they might not agree on someone who they all trust, or maybe they simply don't want to involve anyone they don't have to. It's not immediately obvious that it's possible to do this computation without involving an external party but, as it turns out, it can be done. To see why, let's first observe a simple fact of addition. Let Asher, Henry, and Fang have the following numbers and split them into the following parts.

$$\text{Asher's Secret} = 9 \equiv 2 + 3 + 4 \ (mod\ 37) \tag{1}$$

$$\text{Henry's Secret} = 4 \equiv 1 + 10 + 30 \ (mod\ 37) \tag{2}$$

$$\text{Fang's Secret} = 22 \equiv 6 + 17 + 36 \ (mod\ 37) \tag{3}$$

Now, it's clear to see that if everyone simply added their split numbers back together then we could easily compute the total to be :

$$total = (2+3+4)+(1+10+30)+(6+17+36) \equiv 9+4+22 \equiv 35 \ (mod\ 37) \tag{4}$$

However, if we instead add the columns first, instead of the rows, we see that :

$$\begin{aligned} total =&(2+3+4)+ && (5)\\ &(1+10+30)+ && (6)\\ &(6+17+36) && (7)\\ =&9+30+70 \equiv 35 \ (mod\ 37) && (8)\end{aligned}$$

That is to say, if all the parties involved added the first of their splits together, and did a similar operation with the second and third of their splits, then we could compute the total. This fact relies on the commutativity and associativity of addition. With this, let's devise a simple addition protocol.

Let Asher, Henry, and Fang randomly split their secret into three different parts, called *shares*, in the following manner, where p is a prime number. Because there are three of them, let $p > 3$.

$$\begin{aligned} \text{Asher's Secret} &\equiv \alpha_1+\alpha_2+\alpha_3 \ (mod\ p) && (9)\\ \text{Henry's Secret} &\equiv \kappa_1+\kappa_2+\kappa_3 \ (mod\ p) && (10)\\ \text{Fang's Secret} &\equiv \phi_1+\phi_2+\phi_3 \ (mod\ p) && (11)\end{aligned}$$

An example of how Asher might do this is by first randomly generating α_1 and α_2 and then setting $\alpha_3 \equiv \text{Asher's Secret} - \alpha_1 - \alpha_2 \ (mod\ p)$. Once all three of them generate their shares they now send them in the following manner :

$$\begin{aligned} \text{Asher receives :}&\kappa_2,\kappa_3,\phi_2,\phi_3 && (12)\\ \text{Henry receives :}&\alpha_1,\alpha_3,\phi_1,\phi_3 && (13)\\ \text{Fang receives :}&\alpha_1,\alpha_2,\kappa_1,\kappa_2, && (14)\end{aligned}$$

They now have everything they need to compute the total. They only need to add the corresponding shares together. Each player can calculate two of the three s_1, s_2, s_3 below and once they're finished they'll announce their sums publicly.

$$\begin{aligned} s_1 &= \alpha_1+\kappa_1+\phi_1 = 2+1+6 = 9 && (15)\\ s_2 &= \alpha_2+\kappa_2+\phi_2 = 3+10+17 = 30 && (16)\\ s_3 &= \alpha_3+\kappa_3+\phi_3 = 4+30+36 = 70 && (17)\end{aligned}$$

Once this is done, everyone can simply add these numbers together.

$$total \equiv s_1+s_2+s_3 \ (mod\ p) \equiv 9+30+70 \ (mod\ 37) \equiv 35 \ (mod\ 37) \tag{18}$$

It's clear to see that this could be extended to more players and larger sums, if the prime used to *mod* by were increased. As was mentioned in the introduction, this protocol could also be used as a way to count votes, where a 1 counted as a vote for a candidate, and a 0 counted as a vote against a candidate.

It's easy to verify that, if everyone were to follow the instructions, that this protocol computes the correct result ; namely, that *total* is the sum of the three player's secret number. However, does this protocol maintain privacy ? In this context, this is equivalent to asking if any of the players involved could discover what another player's secret number is.

2 Privacy and Two-Party Computation

SMC protocols were developed in order to replace a trusted external party which could do the computation for us. Thus, when analyzing the security of a protocol, it's natural to compare it to this ideal situation. That is, the intuition for whether a protocol is secure or not is to ask if it's equivalent to interacting with a trusted party. Clearly it won't be exactly the same procedure, as intermediate values must be exchanged between parties to jointly evaluate our function. Thus, we must see if these intermediate values allow players to learn an other player's secrets.

Let's look at the addition protocol given above. The information that a player receives by following the protocol is the *shares* that are passed to them, the intermediate sums s_1, s_2, s_3, and the total. Now, it's assumed that the shares that each player produced at the beginning of the protocol were chosen randomly, and thus, since no player has all three shares of a secret, no one has enough info to reproduce a secret. Clearly each player must learn what the total is by the end of the protocol, so this doesn't in and of itself lead to a memory leak. Thus, the only possibly dangerous information that a player learns is the intermediate sum that they weren't able to compute themselves, as each player can only compute two of the three s_1, s_2, s_3. However, it's possible to argue that this doesn't compromise privacy.

Let's focus on Asher for the following discussion. We've proved above that knowing only s_2 and s_3 doesn't cause a memory leak, but let's assume that Asher magically new what *total* was before the others. Asher could then exactly compute the missing intermediate sum they don't have $s_1 = total - s_2 - s_3$. We can also see that this is the only new value that they could learn by knowing *total*. However, this is the only new information that they learned during the protocol, namely Henry and Fang announce it after they add the shares that were passed to them. So, given what Asher is supposed to learn, namely *total*, we can already compute what he learned by following the protocol, s_1, and thus learning this during the protocol means Asher learns nothing beyond what *total* is. Thus, this protocol maintains privacy. (21).

However, if two people were to collude during the protocol then privacy would no longer hold. This is simple to see. Imagine Asher and Henry worked together to try to find out Fang's secret number. Between the two of them they could construct $\phi_1 + \phi_2 + \phi_3 =$ Fang's Secret. Although this is an interesting topic, this will be outside the scope of this paper. From now on we'll be imposing the restriction that our protocols will only involve two players, and only one of them is mischievously attempting to learn the others secret. This type of SMC is called 2PC (two party computation).

We can now work towards a more rigorous definition of privacy. As stated at the beginning of this section, our protocol is successful if appears that the player's are simply interacting with a trusted party. However, the leaked values interfere with this illusion. Thus, it would seem that "A protocol is private if it always holds that the leaked values can be computed efficiently from the allowed values." With this idea in mind, it's pertinent to think of a program which can compute these leaked values efficiently. We'll call such a program a *simulator* and require that it efficiently takes the allowed values as input and produces the leaked values as output. Thus, privacy can be thoroughly defined as "There exists an efficient simulator such that the simulated values and the leaked values have the same distribution." It's clear that it must be a distribution, as each time the protocol is run, different numbers are chosen randomly, and thus the simulator will have to use internal random choices. Thus, we want the simulator to produce the same probability distribution as the probability distribution of the actual leaked values (21).

We see that we proved this exact scenario regarding our addition protocol, in that Asher only learned s_1. It's clear that the computation was efficient as it only required subtraction. This type of argument is a called a *simulation argument* and is the general way that we can prove protocols maintain privacy.

The analysis above is all specific to our addition protocol, but there exist many other proposed SMC protocols. How do we know that general protocols maintain privacy ? Moreover, we made the assumption that all the players followed the algorithm. It's clear that if one player provided false shares then the resulting computation wouldn't accurately reflect the total sum of everyone's secrets. How could we detect that something like this happened, and how could we protect against it ? This leads us to classifying our adversaries.

3 Adversary Models

Proving security of an SMC protocol requires a carefully defined model. In general, we seek formalizations of the protocol's functionality, network model, adversary, and security type (13). Of particular interest to this discussion is the adversary model. Suppose we are considering an SMC protocol, Φ, among n parties. That is, given some set of inputs $X = \{x_1...x_n\}$, with each input x_i belonging to participant i, Φ attempts to compute $y = f(x_1...x_i)$. For Φ to be correct, each honest party must obtain y ; for Φ to be private, any participant i must gain no additional information apart from y after running Φ.

Proving either correctness or privacy of Φ requires careful attention to the adversary model. Generally, at least some of the n participants are expected to be adversarial. If P is the set of n participants, we may assume there is some subset $C \subseteq P$ of adversarial participants ; these participants are said to be *corrupted*. In the context of this paper we'll be making the assumption that $|P| = 2$ and $|C| = 1$.

We first define the adversarial behavior of a participant i. If i executes protocol Φ as instructed, and does not attempt to uncover any additional information, i is called an *honest* participant. A *semi-honest* adversary will not deviate from the protocol Φ, but will otherwise attempt to gain additional information about the inputs – semi-honest adversaries are 'correct but sneaky'. In particular, semi-honest adversaries may exchange information with other corrupted parties ; in doing so, they may recover information not available to any single participant.

In contrast to semi-honest adversaries, corrupted parties may also be malicious. In addition to the powers of a semi-honest party, malicious parties do not need to follow Φ. A malicious party is free to choose responses to any phase of the protocol in an attempt to disturb protocol correctness or cause information leaks.

Given these two types of adversaries a natural question arises. Which protocols are safe against which types of adversaries, and how can we guarantee that privacy holds ?

It turns out that certain primitives exist which answer these questions.

4 Semi-honest Adversaries and Oblivious Transfer

Before we go into what we need in order to ensure security against Semi-honest adversaries, let's discuss the types of functions that we'll be computing. We're interested in *universal* 2PC (two-party computation) – that is, we wish to show a construction that will work for any computable function f. In order to do this, we consider a circuit representation of f. Any computable function can be represented by a circuit composed of AND and XOR gates (together, these gates are universal). If 2PC is possible on both these gates, it should be possible on any computable function. For simplicity we'll be confining ourselves to bit-wise XOR and AND gates, as we could extend this to being able to share however many bits are needed for the intended computation. Moreover, f should be a one way function. That is, it should be easy to input values to f and receive an output from f, but it must be hard to invert this process and discover what the inputs to f were based on the output. It's clear that f must have this property, otherwise it wouldn't matter if the process to compute f was secure, as a party could simply invert the output to find what the other parties secrets were. With this let's proceed.

As it turns out, we've already covered the XOR circuit, as bit-wise XOR is equivalent to addition modulo 2. Moreover, we also showed that this protocol is secure against one Semi-honest adversary by the simulation argument provided above, and thus it's secure in the 2PC setting. Thus, we only need to construct a Semi-honest secure AND gate and we'll be able to evaluate any computable function securely.

As is well known, AND has the same truth table as bit-wise multiplication, and so constructing an AND gate is equivalent to constructing a protocol that securely computes bit-wise multiplication. If we naively try to use a method similar to addition we see that we run into issues. Namely, let Alice and Bob have inputs a and b respectively. They wish to compute $result = a * b$. If they were to split their secrets into the shares $a = a_1 + a_2$ and $b = b_1 + b_2$ then :

$$result = (a_1 + a_2) * (b_1 + b_2) \tag{19}$$

However, when they distribute the shares as before then Alice has a_1, b_1 and Bob has a_2, b_2. Sadly :

$$result \neq (a_1 + b_1) * (a_2 + b_2) \tag{20}$$

Thus, we need a new primitive to solve this problem. This primitive is called Oblivious Transfer and is defined as follows : Let there be two parties, a sender S and a receiver R. Let S have the bits $\{b_1, b_2, ..., b_n\}$. It's the case that R knows S has n bits, but R only wants to learn one of them, say b_i. Oblivious Transfer is the process by which S correctly sends b_i to R, but S doesn't know which bit in particular that they sent. In this paper we'll assume that there exists a classical implementation of such a protocol. It turns out that one does exist, and it depends on having classically secure public key encryption schemes. The reader is encouraged to read (6) for further details.

With this let's see how we can construct our AND gate. We'll start with the same setup as XOR. Let Alice have a bit a and Bob have a bit b, and let them generate random shares such that $a = a_1 + a_2$ and $b = b_1 + b_2$. Let them send their shares such that Alice has a_1, b_1 and Bob has a_2, b_2. Now is where things change. Let Alice generate a random bit r^1. Then have her prepare the following four states :

$$r^2_{ij} = \begin{cases} r^1 + (a_1 + 0) * (b_1 + 0) & \text{if } i = 0, j = 0 \\ r^1 + (a_1 + 0) * (b_1 + 1) & \text{if } i = 0, j = 1 \\ r^1 + (a_1 + 1) * (b_1 + 0) & \text{if } i = 1, j = 0 \\ r^1 + (a_1 + 1) * (b_1 + 1) & \text{if } i = 1, j = 1 \end{cases}$$

After Alice has done this work, let Bob inspect his shares a_2, b_2. Depending on what they are, have Alice obliviously transfer the corresponding r^2_{ij} above. For example, if $(a_2, b_2) = (1, 0)$ then Alice will send r^2_{10}, or if $(a_2, b_2) = (1, 1)$ then Alice will send r^2_{11}, etc. Once this Oblivious Transfer is complete, Alice announces r^1 and Bob announces r^2_{ij}, and they both compute $\textit{result} = r^1 + r^2_{ij}$. This ends the protocol, as *result* is the product of their two bits a and b. Let's first discuss why the protocol computes the correct result and then discuss why it's secure against a Semi-honest adversary.

It's clear to see that they'll both compute the correct result if we rewrite what r^2_{ij} really represents. Namely, it can be rewritten as $r^2_{ij} = r^1 + (a_1 + a_2) * (b_1 + b_2)$. It will always be this way because Bob's request depends on what his two shares a_2, b_2 are, and he'll always be sure to request the bit that can be rewritten this way. With this, and keeping in mind that for any bit b, $b + b = 0$, we see that :

$$\textit{result} = r^1 + r^2_{ij} = r^1 + [r^1 + (a_1 + a_2) * (b_1 + b_2)] = (a_1 + a_2) * (b_1 + b_2) = a * b \qquad (21)$$

Now that we know we're computing the correct result, let's discuss why this protocol maintains privacy. As a reminder, privacy is the idea that the leaked values can be computed efficiently from the allowed values. Here, the allowed value is simply the result of $a * b$. There are three phases in which security could be compromised : the initial sharing of a_1, a_2, b_1, b_2, the oblivious transfer, and the last transfer of r^1 and r^2_{ij}. Because a_1, a_2, b_1, b_2 are generated randomly, the initial sharing won't compromise the bits a or b. By assumption, it must be that oblivious transfer works as intended and doesn't compromise security, and so the very last transfer of r^1 and r^2_{ij} is our main concern. Well, *result* is an allowed value of the protocol, and it's simple to see that both of them could construct their corresponding leaked values of r_1 and r^2_{ij} using *result*. We also know that nothing else could be leaked from these values as there's no way to unmask what's inside $(a_1 + a_2) * (b_1 + b_2)$, otherwise we wouldn't have needed this protocol in the first place. Thus, this protocol is safe against Semi-honest parties.

With this, we can now XOR and AND two bits securely between two parties, even if one of the parties is Semi-honest. However, we're not safe if one of the parties decides to not follow the protocol. Namely, an adversary could simply feed inputs that don't correspond to what their secret actually is. Thus, we need something to ensure that both parties are following the protocol. If one party were to discover that the other one broke any of the rules, we need a way for them to shut down the protocol and try again.

5 Malicious Parties and Zero-Knowledge Proofs

In order to protect ourselves from a malicious adversary, we need a way to guarantee that both parties are following the protocol correctly without a party having to provide direct proof. If they were forced to provided literal evidence that they were following the protocol then the computation would no longer be private. At first glance it would seem like we're at an impasse. We need proof from the opposing party, but they aren't allowed to show us. It turns out there's a way around this called a Zero-Knowledge proof, and relies upon the idea of indirect measurement.

A zero-knowledge proof allows a prover to convince a verifier of an assertion without revealing information beyond the fact that the assertion is true(25). Let's consider the following simple example of two people, a verifier Victor and a prover Peggy. Let's randomly place Peggy inside of a tunnel. This tunnel will have a left and a right exit to it, and will also have a locked door between these exits, which Peggy claims to possess the key to. Victor, who is outside the tunnel, wants Peggy to prove that they have the key, but Peggy doesn't want to show it to them. Instead, Peggy asks which side of the tunnel Victor would like her to exit from. Thus, by running the experiment multiple times, and Peggy repeatedly exiting on the side that Victor tells her to, it becomes more and more likely that Peggy actually has the key to the door inside the tunnel. In this way, Peggy doesn't need to show Victor the key, but Victor is convinced that it's highly probable that Peggy indeed has it.

Formalizing this concept, we see that three conditions must hold for a Zero knowledge proof : correctness, soundness, and zero-knowledge. In order to talk about these ideas we introduce a Boolean function $\phi : X \to \{0, 1\}$, where any input $w \in X$ such that $\phi(w) = 1$ is called a witness to ϕ. Moreover, we require that ϕ is efficiantly decidable and is polynomially bounded (15). This means that, if we find a witness for ϕ, we can verify that it's a witness in polynomial time. Thus, the goal of a prover is to prove to the verifier that they have a witness to ϕ, and the goal of the verifier is to only believe that the prover has a witness when they actually do. With this formalism, correctness is the property that, if the prover really does have a witness, then an honest verifier will be convinced of it. Soundness is the idea that, if the prover doesn't have a witness, then the honest verifier will only be convinced that the prover has one a small amount of the time. Lastly, zero-knowledge is the idea that, even if the verifier isn't honest, the only thing they learned during the exchange was the strong belief that the prover has a witness.

With this primitive we can now present a powerful theorem : Assume f is a one way function. There exists a general transformation T such that, for any two party function f and for any polytime protocol π that securely realizes f for Semi-honest parties, the protocol $T(\pi)$ securely realizes f for general malicious parties (6). Although this holds for general SMC, we'll only be concerning ourselves with the 2PC case. This transformation T is intuitively simple to understand. Because f is secure against a Semi-honest adversary, we only need to insure that the malicious party follows the protocol. Thus, whenever parties share information, we must include with this message a Zero-Knowledge Proof that the actions performed since the last message were in accordance with the protocol π and would take us from previous state of the protocol to the current state of the protocol. This forces the adversary to follow the protocol, and thus our protocol is secure. In order to formally prove our theorem we need to present one last primitive called a commitment.

In essence, a commitment is a way for someone to agree to a value without having to reveal it. A classic example would be an awards ceremony, where a decision was sealed inside a letter. The judges have *committed* to the decision that's inside, which they can't change, and which no one else knows until the host reveals it. Thus, a commitment involves a *sender* and a *receiver*, and can be broken into a commit phase, and a reveal stage. The commit phase involves the sender taking a bit σ, and sending a commit string c to the receiver as proof of the commitment. The sender will then send a decommit message, call it ρ, to the receiver, which the receiver can use in conjunction with c to obtain σ. Moreover, c must be constructed such that only one value of σ is accepted after decommitment with ρ. This property is called *binding*. It's also required that you can't determine the value of σ from c alone. This property is called *hiding*(15).

With this scheme let's prove the proposed theorem. Let Alice and Bob be the two parties, and let them split their corresponding inputs into shares a_i and b_i for $i \in \{0, 1, ..., n\}$. Then have them send corresponding commitments c_{a_i} and c_{b_i} for each share to each other, along with corresponding decommitment strings ρ_{a_i} and ρ_{b_i}. Thus, we can be sure that we start the protocol correctly, as each share accurately represents the actual inputs that both Alice and Bob started with. Now that we've started the protocol correctly, we want to be sure that we continue to follow the protocol. To do this we need to keep a transcript of what messages have been sent to each other, call it τ. Thus, at the start of the protocol the transcripts looks like :

$$\tau = \{c_{a_i}, \rho_{a_i}, c_{b_i}, \rho_{b_i}\} \tag{22}$$

Now, we require that after each step of the protocol Alice and Bob must send each other messages of what they've done so far. So, after the k^{th} step, they'll send each other corresponding m_k^a and m_k^b messages, where m_k^a represents the k^{th} message that Alice sends to Bob. Thus, we can view the transcript as :

$$\tau = \{c_{a_i}, \rho_{a_i}, c_{b_i}, \rho_{b_i}, m_0^a, m_0^b, m_1^a, m_1^b, ...\} \tag{23}$$

Now, the content of these messages is, given the initial shares a_i and b_i, running the protocol π on those values would produce the messages m_i being sent so far. That is to say,

$$\tau_k = \tau_{k-1} + m_{k-1} \tag{24}$$

Most important to this whole argument is that each party must provide a Zero-Knowledge proof of the equation above. In this way, they can be sure that the step just performed was the correct step in the protocol.

We did brush over one issue. Namely, we assumed that all the values used in the protocol which are required to be generated randomly were indeed randomly generated. In our case these values were the initial shares as well as the decommit strings ρ_i used to commit to these shares. If they aren't generated randomly then it would be possible for the malicious adversary to deduce information they shouldn't be able to. An example of this would be the Oblivious Transfer. Although we didn't go over it's construction, it requires that the receiver has n encryption schemes while only knowing how to decrypt one of them. Thus, if we didn't use truly random numbers to generate these encryption schemes then we could know multiple decryptions, which would allow us to gain more information from the sender than we should. Luckily, there's a solution. Let Alice produce a random tape RT_a which has all of the random bits that they'll need to complete the protocol at the very beginning, call them t_{a_i}, and then have her send a commitment to these random values to Bob. Then have Bob decommit these bits, and randomly generate just as many bits as Alice has, call them r_{b_i}. Then have Bob construct the values :

$$s_{a_i} = t_{a_i} + r_{b_i} \tag{25}$$

Have him send these values back to Alice. Alice must now replace her random tape RT_a with the bits s_{a_i}. Thus, Alice's tape will at least be just as random as r_{b_i}. Repeat a corresponding process for Bob's random tape RT_b. With this, we can guarantee the random bits are at least as random as the honest party's random tape, which is the best we can hope for. Thus, the fully correct transcript looks like :

$$\tau = \{c_{a_i}, \rho_{a_i}, c_{b_i}, \rho_{b_i}, RT_a, \rho_{RT_a}, RT_b, \rho_{RT_b}, s_{a_i}, s_{b_i}, m_0^a, m_0^b, m_1^a, m_1^b, ...\} \tag{26}$$

With this we can ensure that our Semi-honest protocols are safe against malicious parties, just as required.

This concludes our discussion of classical 2PC. It turns out there exists extensions of these ideas which cover general SMC, but that is beyond the scope of this paper. It shouldn't be too far of a stretch though, as the addition protocol presented above is multi-party and secure. All that's left to do is devise a multiparty multiplication protocol, and then a compilation from Semi-honest to malicious parties as we just did.

We now look at how the landscape changes when we move from the classical to the quantum world.

Quantum SMC

So far, we've assumed that every part of our protocol is classical. We've been performing classical computations on classical data with classical adversaries. What happens when we take this to the quantum setting? Two immediate issues are the fact that we can't clone quantum data, and that data vanishes as a protocol evolves through measurement. So we start asking ourselves, can we implement our classical primitives using quantum gates and expect the same behavior ? If not, why ? What must we change to fix these possible issues ? Also, is it reasonable to only consider Semi-honest and malicious adversaries, or are there more appropriate definitions now that we've entered the quantum setting ? Does the idea of having a single simulator in our definition of privacy still make sense, or do we need multiple simulators due to not being able to rewind our state ? Does there exist a finite number of universal quantum gates that encapsulate quantum computations that we can show security for such as XOR and AND in the classical setting ? If we could possibly define and prove quantum security, is it possible to compose two such protocols in a way that the result is also secure ? Finally, if an adversary has access to a quantum computer, will our classical protocols still be secure ? While all of these questions are interesting, the last two questions will be the focus of our paper.

We'll continue to focus on 2PC, and show that work by Hallgren, Smith, and Song suggests that classical protocols are indeed able to achieve computational security against a quantum attacker (22). In this section, we outline their constructions for showing quantum security of classical protocols.

6 Stand-Alone vs. Universal-Composable Secure Computation

There are two important families of secure protocols : stand-alone protocols (19) and universal-composable (UC) protocols (20). Both frameworks seek to 'modularize' the security of a protocol, that is, they allow a protocol to prove security given some ideal functionality, and remain secure when that function is replaced with another composable protocol. Broadly, we wish to capture the idea of *emulation*, defined in (22) as follows : suppose we have two protocols, $\varPi$ and $\varGamma$. $\varPi$ 'emulates' $\varGamma$ if, for any attackers A and S, the machines made by composing the attacker and the protocol, $M_{\varPi,S}$ and $M_{\varGamma,A}$, are indistinguishable. Hallgren et al. extend both models to fit a quantum setting.

In the stand-alone setting, this can be formalized concisely. Suppose we wish to achieve some ideal function f. In the ideal world, f would be carried out by a trusted party. However, in the real world, we must make do with the protocol π, which is not trusted. We say that π *securely evaluates* or *realizes* f if, for any adversary A, there exists an adversary S such that the output of π with adversary A is indistinguishable from f run with adversary S. In other words, no adversary can do better against the real protocol, π, than the ideal function f. (4) This allows proofs to be carried out assuming an ideal functionality, which may then be 'swapped out' for a protocol that realizes it.

In their formalization, Hallgren et al. include an *environment machine*, Z, which is only interacted with at the beginning and end of the protocol. Z may be split into two machines, Z_1, and Z_2, where Z_1 samples the protocol input (and possibly additional information). Z_2 samples the output state, and attempts to decide whether it is interacting with a protocol implementation π, or the ideal functionality f. Abilities of the attacker are in part captured by Z ; for example, Z may be allowed to carry state from Z_1 to Z_2 (as opposed to making a decision based purely on the machine input), and a computational bound may be imposed on Z. Quantum abilities of the attacker are captured by the notion of a quantum advice string, which may be provided to Z_1, Z_2, or both.

The UC framework extends the power of the environment. The environment should now capture everything that can happen outside the protocol, including other protocol executions, adversaries, and human users. In order to achieve this, the environment is made interactive – the adversary can now interact arbitrarily with the environment at any step of the protocol. The UC model is therefore stronger than the stand-alone, since the adversary and environment may now have more opportunities to interfere with or attack the protocol. If π is indistinguishable from f in the UC setting, it is said to UC-realize f. An important property of UC-secure protocols is *universal composition*, which allows a protocol to remain secure even if run concurrently with an unbounded number of other protocols. (4)

Hallgren et al. give a quantum UC construction as well. Similar to the standalone setting, this simply allows the environment to take quantum advice at each step.

7 Proving Quantum UC Security in the F_{zk}-Hybrid Model

A major contribution of (22) is proving the UC-security of a classical protocol given access to a secure zero-knowledge proof of knowledge, F_{zk}. F_{zk} is defined as follows : suppose we have two communicating parties, Alice and Bob, and an NP relation, R_L. F_{zk} allows Bob to verify messages from Alice. When Bob receives (x, w) from Alice, F_{zk} verifies whether (x, w) is in R_L. If it is, the message x is forwarded to Bob, otherwise Bob is instructed to abort.

Before continuing, it is important to acknowledge the limitations and assumptions of proof made by Hallgren et al. First, quantum must be polynomially-bounded and statically-corrupting. Second, the proof requires the existence of a pseudorandom generator (PRG) secure against a quantum attacker, and a dense classical public-key cryptosystem that is IND-CPA secure against a quantum distinguisher.

To show UC security, (22) develop an abstraction called the *simple hybrid argument* (not to be confused with the 'hybrid model'). This technique stems from techniques used in classical cryptography to prove the indisinguishability of two distributions. Essentially, a classical hybrid argument states that the difference between two distributions D_1 and D_2 can be split into a sum of differences between intermediate distributions. Formally, it states that we may create t hybrid distributions, denoted $H_1, H_2, ..., H_t$, such that, for any efficient attacker A, $\sum_{i=1}^{t-1} Adv_A(H_i, H_{i+1}) = Adv_A(D_1, D_2)$. Here, $Adv_A(X_1, X_2)$ indicates the 'advantage' of attacker A in distinguishing X_1 and X_2 ; that is, the ability of A to distinguish whether a random element is from either set X_1 or set X_2. The simple hybrid argument used by Hallgren et al. applies a similar logic to the distinguishability of computational machines, i.e., that the (t, ϵ) relationship between two machines M_0 and M_l may be split up between l intermediate hybrid machines ; with t and ϵ indicating the complexity and success chance of a distinguisher respectively.

The simple hybrid argument described above is used to prove security of the CLOS protocol, a previous UC-secure classical protocol (5). Complete description of CLOS is outside the scope of this paper. However, (22)'s argument hinges on the idea that the security of CLOS reduces to simple hybrid arguments against static attackers, where $t = poly(n)$ and $\epsilon = negl(n)$. Hallgren et al. argue that this proof still holds up in a quantum setting, so long as the the assumptions of quantum-secure encryption and random-number generation hold – since the underlying distributions are assumed to made secure against a quantum attacker (22). CLOS's security additionally requires the 'commit-and-prove' functionality F_{cp}, but (22) argues that this is also achievable given a secure PRG and secure oblivious transfer constructed from the public-key cryptosystem.

8 Creating a Stand-alone Secure F_{zk} Realization

So far, we have described how Hallgren et al. prove security of a UC-secure shared protocol. However, this proof was executed in the F_{zk}-hybrid model – that is, it assumes access to quantum-secure zero-knowledge arguments of knowledge (ZKAoK). Therefore, (22) also show the construction of a stand-alone ZKAoK protocol, Π_{zk}. Similar to the previous section, this construction assumes access to a quantum-secure public key dense encryption scheme.

Similar to classical zero-knowledge, the goal is to design a protocol allowing the proving party to prove something to the verifier without revealing additional information. This is modeled via a *witness* w, which the prover may see, but should be kept secret from the verifier. Zero knowledge is shown by a simulator argument – that there must exist a simulator, S, that can always replicate the verifiers output, but is guaranteed never to see w.

In addition to zero-knowledge, the protocol requires *proof of knowledge*, that it, is must guarantee that the prover actually has w (the 'argument of knowledge' referred to in (22) is a similar notion, but applied to a polynomially-bounded attacker). This is modeled by the *extraction* property. Extraction states that there must exist some extractor machine E, which is able to recover w given oracle access to the prover. Extraction and zero-knowledge would seem to be mutually exclusive ; however, they are not. Normally, extraction is proved using a rewinding argument, in which the extractor is allowed to view outputs of the prover oracle, then rewind the prover oracle to its previous state. Now empowered with knowledge of the prover's future outputs, the extractor can carefully choose its messages to force the prover to give up w. This allows a proof-of-knowledge, but does not necessarily break zero-knowledge (since the verifier is assumed to be unable to rewind the prover in the real world).

However, in the quantum realm, the (possibly malicious) verifier is modeled by a quantum machine. This complicates proofs of zero-knowledge proof-of-knowledge, since such proofs often rely on rewinding techniques. Rewinding generally relies on saving the state of a machine and returning to it later. This is easy for a classical machine, but quantum states cannot be cloned – meaning it may not be possible to rewind a quantum machine.

Fortunately, work by John Watrous in (24) lays a groundwork for proving zero knowledge against a quantum verifier, which Hallgren et al. extend into their proof of security for the protocol Π_{zk}. (24) proves that quantum-secure zero-knowledge protocols can be constructed for any language in NP, assuming the existence of quantum-secure commitment schemes. (22) uses this construction to implement F_{zk}.

Essentially, Π_{zk} begins with a 'coin flipping' preamble, allowing the prover and verifier to exchange random strings, a and b. These strings may then be XOR'd to create a public key, pk, for the assumed public-key encryption scheme. Next, the prover sends an instance x of an NP language, and the encryption of the witness w, $e(w)$, to the verifier. Watrous's zero-knowledge protocol in (24) allows the prover to show that w is indeed a witness of x, without actually being able to decrypt $e(w)$.

Extraction is provided by the construction of pk. Recall that pk was constructed by random strings a and b, where a is chosen by the prover, and b by the verifier. The intention is that pk is a public key for which the corresponding secret key, sk, is unknown. Because the prover can always send a random string, this is very likely to be the case. However, in an extraction setting, the verifier is allowed to observe a before deciding on b; given this foreknowledge, b can be chosen to yield pk such that the secret key is known to the extractor. The extractor can then simply decrpyt $e(w)$.

Thus, it is possible to create a quantum-secure ZKAoK protocol – meaning the CLOS protocol can be made quantum-secure as described above.

Conclusion

In this paper we explored the basic construction of classical SMC and the security of 2PC protocols against a quantum adversary. SMC is a thriving field of research in cryptography, and has important real-world applications. The research covered in this paper shows that – given quantum-secure encryption primitives – classical SMC protocols are likely remain secure in the face of quantum adversaries. This is a very encouraging result for the future of private communication in a post-quantum world.

Additionally, the proofs given in (22) give valuable insight into the nature of proving post-quantum security. Quantum machines have fundamentally different properties from classical ones (e.g., no-cloning), meaning that some classical proof techniques may fail in the face of a quantum adversary. There is much valuable work to be done in extending cryptographic proof techniques to the quantum world.

Bibliographie

[1] Aharonov, D., and Ben-Or, M. Fault tolerant quantum computation with constant rate error. In *Proceedings of STOC '97* (1997).

[2] Bell, J. On the Einstein Podolsky Rosen paradox. *Physics 1*, 3 (1964), 195–200.

[3] Bennet, C. H., and Brassard, G. Quantum cryptography : Public key distribution and coin tossing. In *Proceedings of IEEE International Conference on Computers, Systems, and Signal Processing* (1984), vol. 175.

[4] Canetti, R. Universally composable security : A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067, 2000. `http://eprint.iacr.org/2000/067`.

[5] Canetti, R., Lindell, Y., Ostrovsky, R., and Sahai, A. Universally composable two-party and multi-party secure computation. Cryptology ePrint Archive, Report 2002/140, 2002. `http://eprint.iacr.org/2002/140`.

[6] Canetti, R., and Rosen, A. General secure two party and multi party computation.

[7] Claude Crepeau, Daniel Gottesman, A. S. Secure multi-party quantum computation. In *Proceedings of STOC 2002* (2002).

[8] David Chaum, Claude Crepeau, I. D. Multiparty unconditionally secure protocols (extended abstract).

[9] Einstein, A., Podolsky, B., and Rosen, N. Can quantum-mechanical description of physical reality be considered complete ? *Physical Review 47* (1935), 777–780.

[10] Frederic Dupuis, Jesper Buus Nielsen, L. S. Secure two-party quantum evaluation of unitaries against specious adversaries. *Advances in Cryptology* (2010).

[11] Frederic Dupuis, Jesper Buus Nielsen, L. S. Actively secure two-party evaluation of any quantum operation. In *Proc. CRYPTO 2012* (2012).

[12] Greenberger, D., Horne, M., and Zeilinger, A. Going beyond Bell's theorem. In *Bell's Theorem, Quantum Theory and Conceptions of the Universe* (1989), M. Kafatos, Ed., Kluwer Academic Publishers, pp. 69–72.

[13] Ishai, Y. Secure multiparty computation i. Cryptography Boot Camp lecture, 2015.

[14] Kaushik Chkraborty, A. L. Practical position-based cryptography. *Physical Review A* (2015).

[15] Lindell, Y. Tutorial on secure multi-party computation. T.J. Watson.

[16] Mermin, N. *Boojums all the way through.* Cambridge University Press, 1990.

[17] Michael Ben-Or, Claude Crepeau, D. G. A. H. A. S. Secure multiparty quantum computation with (only) a strict honest majority. In *Proceedings of FOCS '06* (2006).

[18] N. Chandran, V. Goyal, R. M. R. O. Position based cryptography. In *Proceedings of CRYPTO 2009* (2009).

[19] Oded Goldreich, Silvio Micali, A. W. How to play any mental game or a completeness theorem for protocols with honest majority. *STOC* (1987).

[20] Ran Canetti, Yehuda Lindell, R. O. A. S. Universally composable two-party and multi-party secure computation. *STOC* (2002).

[21] Ronald Cramer, Ivan Bjerre Damgard, J. B. N. *Secure Multiparty Computation and Secret Sharing.* Cambridge University Press, 2015.

[22] Sean Hallgren, Adam Smith, F. S. Classical cryptographic protocols in a quantum world. *International Journal of Quantum Information* (2015).

[23] SMITH, A. Cryptographic protocols in a quantum world. Proceedings of PIRSA-2007, 2007.

[24] Watrous, J. Zero-knowledge against quantum attacks.

[25] Yuval Ishai, Eyal Kushilevitz, R. O. A. S. Zero-knowledge from secure multiparty computation. In *Proceedings of STOC-2007* (2007).