Crypto Final Project: Analysis of vulnerabilities and mitigations in Multiparty Computation

Leor Fishman

Acronyms used in this paper:

(s) MPC: (secure) Multiparty computation

SFE: Secure Function Evaluation

SPDZ: a particular protocol for sMPC focused on increasing computational speed and decreasing communications overhead, involving an offline processing phase and an online computational phase. Achieves linear time multiplication via a process known as triple generation.

OTP: One time pad

XOR: bitwise exclusive or

MPSS: Multiparty Secret Sharing, a high-linear-time overhead protocol for creating and distributing new shares to a given secret without ever reconstructing it.

SSS: Shamir's Secret Sharing, a schema for k of n threshold secret sharing based on polynomial interpolation.

Part 1: Covert protocols and steganographic transfer:

Section I: Motivation:

MPC, or Multi-party computation, is thought by many to be the 'holy grail' of cryptography -- after all, it allows for the secure evaluation of an arbitrary function by any bounded number of parties without revealing anything or exposing attack surfaces but inputs and outputs, against even malicious adversaries. This combination of perfect reliability and perfect secrecy seems to cover everything one needs out of a cryptographic schema. However, while this enforces *computational* privacy -- nobody performing the protocol can learn anything they wouldn't have learned from their inputs and the outputs of the function being evaluated, MPC doesn't yet enforce *personal* privacy -- i.e., MPC protocols do not hide *who* is performing them.

As MPC has developed as a field, many people have tried to create secure mechanisms for performing secure function evaluation *covertly*: in other words, doing SFE/MPC in such a way that none of the parties involved can identify one another or out one another as participating in the protocol. There have been major advancements in the field, e.g. https://eprint.iacr.org/2012/642.pdf which provided a version of covert sMPC interfaced with the SPDZ protocol that provided covert security with SPDZ level performance. (For a more full exploration of covert mpc in general, see https://www.cs.cmu.edu/~goyal/covert-proc.pdf). Now, this is of course an extremely positive development, but it is incomplete. The above proofs demonstrate that it is possible to generate a method for MPC in which adversaries cannot figure out who the honest parties are in computing a protocol. However, consider the secondary threat model in which adversaries know they are in a protocol and are attempting to figure out who else in that protocol is adversarial and communicate their data to one another (this would be relevant, for example, in a case where adversaries have many fragments of a piece of Shamir's secret shared data and are trying to collect their shared information).

Section II: Analysis:

Assume for the moment that we are in an ideal world where all communications between adversaries are monitored and if any non-protocol communications happen, the adversaries have their data deleted, and that data deletion is guaranteed at the end of the protocol as well. A priori, under this Orwellian ideal world, it doesn't seem impossible to prevent adversarial data transfer. However, as I shall show below, under certain additional assumptions, there is in fact no way to enforce this sort of *consensual* covertness. In other words, it is not possible to prevent adversaries from identifying one another or from transmitting data to one another, even under covert MPC protocols.

Before diving into our proofs of noncommunication, I must first establish our security definitions for the two problems being analyzed: Adversaries identifying one another, and adversaries transferring data to one another.

Game 1: Adversary identification.

A random bit B is selected from {0, 1}

The adversary A is involved in a k party protocol which contains a target A* to evaluate. If B = 0, A* is honest and follows the correct protocol. If B=1, A* is dishonest and follows the adversary A's protocol. All other parties are honest. The protocol involves at least one exchange of messages, including at least n bits of randomness (for n = message length), between A and A*.

At the conclusion of the protocol, A guesses a bit b', and wins if b' = b *and* the outputs of A and A' to the protocol are computationally indistinguishable from those of honest parties.

Game 2: Information transmission:

A random message M is selected from {0, 1}^n

The adversary A is involved in a k party protocol which contains a target A* to evaluate. A* follows the adversary's protocol with data M.

At the conclusion of the protocol, A guesses a message M*, and wins if M = M* *and* the outputs of A and A' to the protocol are computationally indistinguishable from those of honest parties.

Theorem I: under the constraint that the protocol in question involves an exchange of randomness between A and A' and that adversaries are allowed to communicate before the shares are distributed and groups are allocated, there exists an adversary A which succeeds at this Adversary Identification game for any such protocol with probability (1 - negl(n)).

Proof: I provide a simple protocol Covert_Handshake that works for any such game.

Step 1: Before the game, generate a polynomial set of random n bit strings P_adv and share them among all adversaries.

Step 2: When the protocol requires a coin-tossing protocol between two parties, then have both randomnesses those parties use be uniformly randomly selected from the polynomial-size subset.
Step 3: A will guess b’ = 1 iff the randomness he receives is in P_adv.

Success of identification: If A* is not an adversary, the randomness A receives will only be in P_adv with negligible probability.

Indistinguishability: In both cases the parties are transmitting random bitstrings back and forth.

Therefore this protocol succeeds at the Adversary Identification game.

Note that this method can be modified slightly to allow for a second protocol, Covert_TransmissionAndHandshake, that succeeds at the Information Transmission game. The protocol is as follows.

Step 1: Before the game, generate a polynomial set of random n bit strings P_adv and share them among all adversaries. Additionally generate one last random n bit string OTP, sharing it among all adversaries (for K-round protocols, generate K OTPs)
Step 2: When the protocol requires a coin-tossing protocol between two parties, then the protocol will go as follows:
T sends a hash of an n in P_adv as his randomness
A tests by exhaustion whether the randomness he received is in P_adv, and if so sends OTP XOR M_A = m’ and otherwise sends a random string.
Step 3: T guesses m’ XOR OTP.

Theorem II: The Covert_TransmissionAndHandshake protocol succeeds with 1-negl(n) probability at the Information Transmission game.

Proof:
Transmission of Information: Assuming A is an adversary, T’s guess will always be equal to M_A XOR OTP XOR OTP = M_A.
Indistinguishability: The back and forth messages are either a random string or a message XOR a random string (indistinguishable from random) and are thus indistinguishable from proper protocol.

Section III: Final Thoughts:

Now, it should be noted that this doesn’t make truly covert protocols impossible -- as noted before the covert handshake protocol, these two methods rely on an exchange of randomness between the adversary and the target. If the adversary and target are both getting their randomness from some external source such as a public random beacon (e.g. that made by NIST, or by a separate set of nodes on the same network as the current protocol) then the steganographic channel which allowed for identity and data transmission is closed off.

Additionally, if use of a public beacon is inadvisable for security reasons (e.g. backdoorability and/or wanting to keep the network's activity private), then the transmission protocol can be closed off by generating randomness before distributing data to all involved parties. This method, however, leaves Covert_Handshake available and thus doesn't preserve covertness against adversaries.

In conclusion, assuming there exists transmission of randomness between parties in a given multi-party protocol, it is not possible to provide covertness between consenting adversaries, even with the ability to terminate the protocol under improper communications. (Equivalently, it is impossible to determine if two people are colluding to reveal their participation in a covert protocol).

Part 2: Analysis of Quorum-based MPC optimizations, particularly the need for MPSS:

Section I: Motivation:

The second piece of this project will be focusing on analyzing the use of constant-size, log-depth groups for sMPC, particularly on vulnerabilities in the method and on ways to mitigate those vulnerabilities. Before diving into those vulnerabilities, it is first necessary to describe in general the model I will be analyzing. Specifically, I will be looking at the Enigma Protocol (specifically the MPC form) for computing over distributed data. This model is generally described here: https://enigma.co/enigma_full.pdf, with a more in-depth, technical description found here: https://dspace.mit.edu/bitstream/handle/1721.1/105933/964695278-MIT.pdf.

In short, the method works as follows: given some piece of data shared over a number of parties (under SSS or similar methods), instead of having those parties execute all computations as one large group, instead divide those parties into many groups of constant size called quorums and have them execute each round of computations individually (a round being defined as all additions until a multiplication, or all multiplications until an addition). Quorums then send their results as further shares to a smaller number of quora in the next round, resolving down to a final 'result' quorum after log(n) rounds. This avoids the quadratic overhead of degree reduction for multiplication, instead replacing it with a logarithmic number of constant-time degree reductions (while avoiding the potentially costly triple generations of SPDZ). A reasonable amount of analysis of this new method was done in the paper itself, focusing on using MPSS (a method for resharing a secret in the MPC model) and a relatively high reshare rate to keep the secret secure. The paper showed that, assuming the transfers of partly completed computations was done with MPSS every round, that no additional information was revealed to any adversaries whether malicious or honest but curious. Additionally, they suggested that this MPSS could be done with a relatively low frequency without reducing the protocol's security, merely enough to avoid adaptive adversarial corruption. However, the paper in particular focused on one and only one threat model: Adversaries controlling enough of one specific quorum to recover all the data (by controlling enough shards to reconstruct it). And, as they argue, with large enough quorum size and strict enough conditions for the algorithm to not abort, the likelihood for this event to happen is near-0 and it isn't a serious concern. Additionally, though they don't explicitly mention this, their system avoids the problem of adaptive adversaries by having a high enough reshare rate -- if you reshare more often than the (fraction of nodes to reconstruct the secret / adversary's power), adaptive adversaries literally cannot retrieve enough information to restore the data. Therefore, I agree that these two threats are not serious concerns.

However, there does exist another type of attack that the paper doesn't mention (or, rather, didn't explicitly consider, choosing instead to implicitly mention a potential solution via MPSS, as shall be elaborated upon) -- that of Gaussian elimination by successive computations. Here is a high-level description of the attack: consider wlog a single quorum of size k in the first round (i.e. only computing sums) of a computation with some number of adversaries. Consider wlog that the quorum is simply computing the summation of shares operation sigma (this is a relatively useless operation in the real world, but simulates the general case of 'additive operations over shares' which would be used in e.g. an averaging operation over many separate data points). Now, note that as part of the operation, unless the (relative to the simple sum operation) costly MPSS protocol is performed after each round, at each round the quorum learns the results of its round (unless it is an output round). For the simple summation case, in the ideal world (i.e. the world where all shares are computed upon simultaneously without quora), this leaks nothing more than publicizing/revealing the final result, which for a large-scale sum operation gives away relatively little information. However, for the real protocol, for each quorum, the adversaries will learn the sum of their quorum -- thus, given that they know their shares, they will learn the sum of the shares that are not theirs as well -- now, in an environment with N shares, adversary power a and quora of size k, after one round the adversary only learns O(N*a/k) equations, and the secret remains secure (barring cases where the adversary controls all but one node in a quorum and thus learns the final share directly, or the already mentioned case where the adversary controls enough of the multiplication quorum nodes to flat-out reconstruct the secret). However, consider many computations over the *same* data, with randomized groups in each case -- intuitively, after each set of these rounds the adversary learns ~O(N*a/k) new equations, and thus after O(k/a) rounds the adversary has N equations for N-N*a unknown shares and can reconstruct every share, and thus the secret. Now, there are a lot of pieces that must be elaborated upon here. First, it is important to note that if MPSS is done after every round to transmit the results of a given computation then this attack is no longer possible -- the adversaries no longer learn intermediate results and thus have nothing to build on for Gaussian elimination. However, as stated above, MPSS is relatively expensive, and ideally would be performed as infrequently as possible. This attack provides a lower bound on the necessary frequency of that MPSS.

Section II: Analysis and results:

Now, rather than attempting to directly calculate closed-form solutions for that lower bound directly, I instead built a randomized simulator of the model, based on the summation operation. The simulator worked as follows: given the number of shares, number of shares needed to reconstruct the secret, the size of the group, and the adversary's power, it would construct a simplified secret network in which every secret was represented by an integer held by a specific node, and nodes were randomly assigned to adversary or honest based on the power stat fed in. The network would then proceed to simulate computational rounds until the adversary held enough unique sums to reconstruct the secret, averaging out that number over a number of trials. An implementation of this simulator can be found at https://github.com/FishmanL/crypto.

Before getting into the results, I should first establish a few things about the simulator. 1) this simulator is purely by checking number of equations, not via a CAS. In other words, it might be possible to recover many more shares in fewer rounds merely by using pseudonymous IDs to set up equations that reveal those shares (e.g. in the simplified 4 share system, imagine learning a + b, a + c, and b + c--even without n equations, you've learned a fraction of the shares). In other words, this is a lower bound on the necessary frequency of reshares, not a flat value. 2) this simulator assumes only static adversaries rather than allowing for adversaries to specifically corrupt those nodes they still need after 2 or 3 rounds. This also points to it's lower bounding nature (in other words, the results section will give certain values for the average length of time until a secret is compromised--one should definitely be using MPSS or resharing significantly *more* often than the graphs would indicate, and certainly not *less.*).

Results: the results of the simulations are summarized in the below graphs:

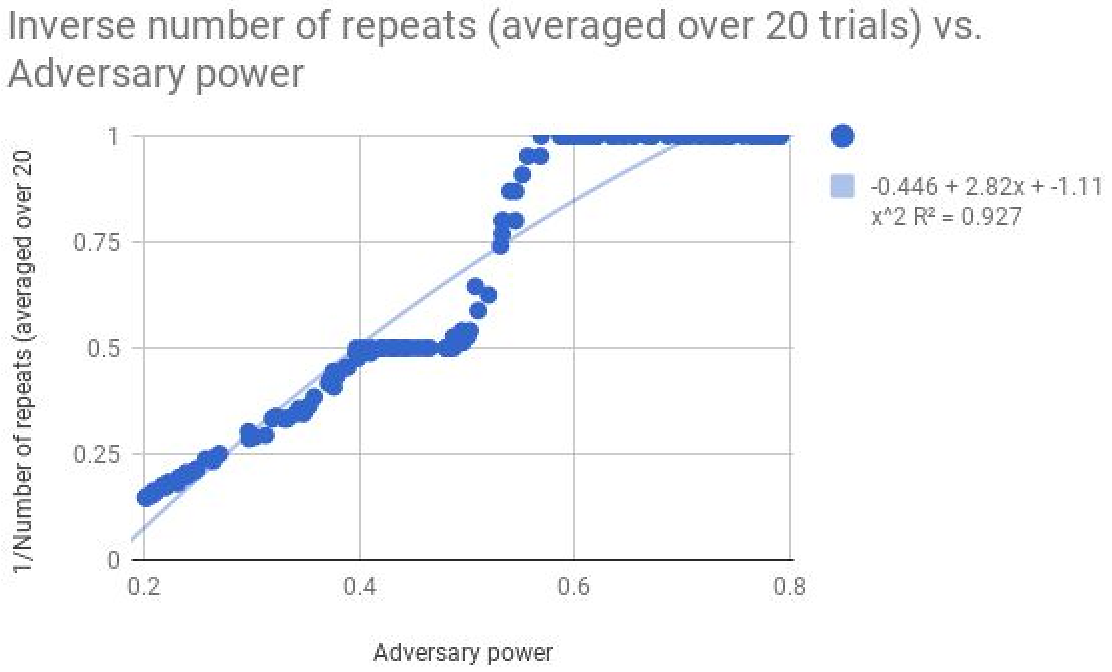

Figure 1: 1/number of trials to break security vs. adversary power, 0.2 - 0.8 advpower.

This graph was produced using the linked github simulator, with groupsize set to 5 (I will discuss the effects of changing group size below), with raw data available here: https://docs.google.com/spreadsheets/d/1-8mRoqjDtsFLeqeimkzMs9JcY7HXoxSkrM83LX76p3w/edit#gid=0 .

Now, the results require a bit of explanation: there are, as can be seen, 4 major slopes in the inverse graph: the period from 0.2 - 0.4, the period from 0.4 - 0.5, the period from 0.5 - 0.6, and the remainder. These 4 states correspond to 4 potential types of computational environments: the first case is when the adversaries have a relatively small fraction of the nodes, and thus on average only have 0 or 1 node per computational group -- in this case, raising the fraction of adversary power increases the number of groups adversaries are in approximately linearly, and thus the number of rounds required to recover all shares goes down inversely to that linearity. The second case is when adversaries on average have 1-2 nodes in each group already, and thus get no new information from a slightly higher frequency of adversarial nodes--they get no more info from having 3 nodes in a group of size five than they would from having two in a group of size 5. The third case is when the frequency of adversary control is high enough that adversaries begin to control all but one of many smaller quora, thus retrieving the specific shares of the final quorum--repeated enough times, this allows for reconstruction of the secret earlier than would otherwise be possible. The final case is simply when on average, the adversaries learn all of the remaining secrets via all-but-one reconstructions after a single round.

The relevant case for our purposes is the first period of shallow inverse slope, and the relevant numbers are that, after a low single-digit number of computations without MPSS, the adversary can generally retrieve all shares and reconstruct the data.

Now, as explained above, with adversary power held constant, the group size should have approximately a linear effect on the number of rounds it takes the adversary to reconstruct all data. Results consonant with this intuitive belief are given below:

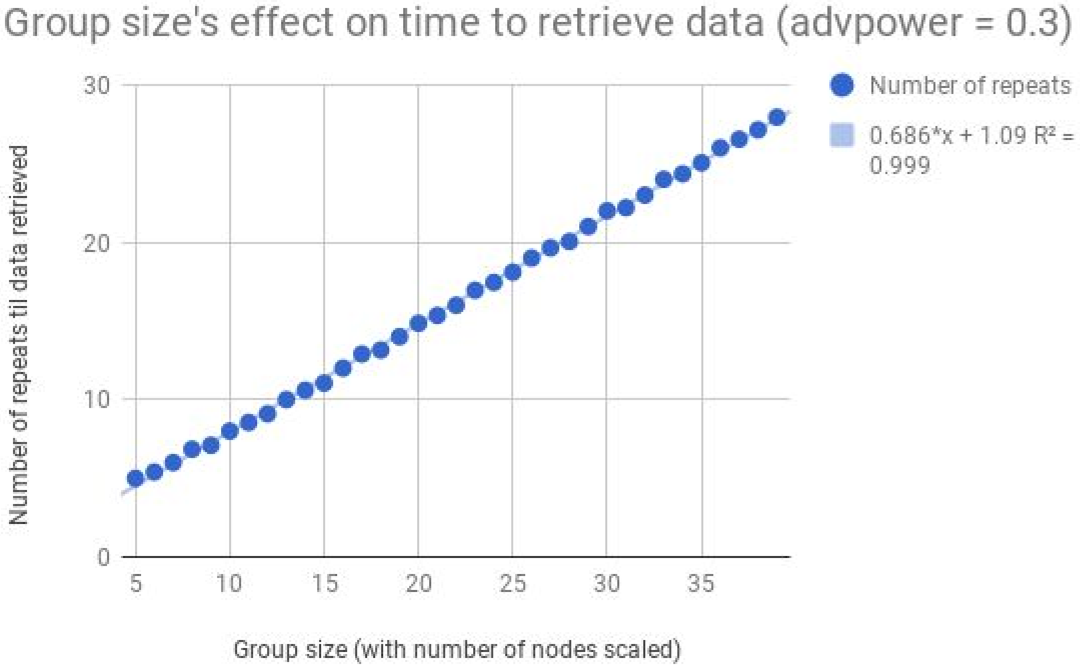

Figure 2: Effect of group size on time to retrieve data.

This shows a linear relationship of group size to number of rounds (the data for this graph is found on the previously linked spreadsheet -- the number of nodes was found to have no effect on time to break security, which is what was expected), which is definitely a plus for the protocol -- as increasing group size will increase security against this sort of attack.

However, note that in order to preserve the protocol's positive performance characteristics, quorum size k must be far smaller than the overall number of shards needed to reconstruct the secret, otherwise the network's runtime will again be dominated by quadratic degree reduction steps. I.e. the group size can only increase the number of rounds to some larger but still constant and finite number of computations.

Section III: Final Thoughts:

Now that the results of these simulations have been discussed, I can turn to discussing my overall conclusions. As mentioned a few times above, MPSS when repeated often enough prevents this attack -- however, it should be emphasized that 'often enough' is based on a number of computations, *not* based on timestep. It should also again be reiterated that these simulations provide a *lower bound* on the frequency of needed MPSS, not a flat number. Adversaries with a CAS will likely be able to learn much more by e.g. finding m < n equations for m of the unknown shares and solving for those shares, potentially recovering the overall data earlier than this simplistic simulation method would suggest.   Now, beyond MPSS, it might be possible to protect against this sort of attack another way: via use of the Laplacian noise formulation of Differential Privacy, a technique involving adding small noise terms to any computation to minimize the information adversaries get about newly added data points and the information leaked by successive computations on the same data. Specifically, if instead of adding the noise at the end of the computation, the noise was computed as part of the protocol and added at the end of each quorum's computations, the Gaussian attack would no longer be possible (the magnitude of the noise term would, of course, have to be downshifted slightly to account for the fact that there are now log(m) noise terms being added for m = size of the computation). The reason I propose this possibility rather than simply sticking to MPSS is computational--this modified operation would only cost as much as a single multi-party addition operation, significantly less than even the one-round optimized MPSS described in the Enigma papers. However, my knowledge about Differential Privacy isn't enough to propose this system as more than a potentiality.