1 The Millionaire Problem

We can create a function just using sums and products in $\mathbb{Z}_2$ that computes the outputs of any circuit, without leaking information about the inputs. For example, imagine that Alice and Bob are two millionaires. They want to learn who is richer, but they don't want to leak information about their own wealth, beyond what can be learned by the output. This is Yao's two-millionaires problem. A practical, but less colorful, example of when we might want to solve this type of problem is when two (or more) people are bidding in an auction. We want to know who has the highest bid, but don't want to leak information about the bids. This information about the highest bidder can then be used in another computation that completes the auction.

2 Secure MPC

The secure multiparty computation problem (MPC) generalizes the 2 Millionaire problem. Consider an *ideal model* with a trusted party who knows a function f over n inputs. Each of the n players sends their input to this trusted party. This party then computes the function f over the inputs and then sends the output to all the players. In MPC, we seek to get the same security guarantees as in this ideal model, but without the trusted external party. In particular, we do not want to leak any information about the inputs, besides what can be learned from the output of f. The function f can be arbitrary; it can be: maxarg, min, a second-price auction bid computation, a market-matching function that matches buyers to sellers based on bid and ask prices[1], etc.

Basically, *any* computation that can be done with a trusted external party can be encapsulated in the function f that is run by this external party over the inputs they receive. Then using secure MPC, the players themselves can just compute f over their inputs, without need of the trusted external party.

3 MPC vs Smart Contracts

MPC is related to but not the same as smart contracts. Some strengths of MPC when compared to smart contracts:

- **Enhanced privacy.** In smart contracts, miners can manipulate transaction order for personal gain after observing all transactions sent out in the P2P network. This problem of front-running and *Minor Extractable Value (MEV)* is both well-known [3, 4] and also widespread [6]. This does not happen with MPC
- **Computation via Participants.** In smart contracts, contract computation is performed by the miners, who may have different incentives than the participants in the contract (cf. the Verifier's Dilemma [5, 7]). In MPC the computation is done directly by the participants in the smart-contract, whose incentives may be more directly aligned with correct computation of the output.

Some weaknesses of MPC when compared to smart contracts:

- **Connection to the Blockchain.** Smart contracts directly connect to the blockchain, and can immediately move cryptocurrency based on the contract output. In contrast, MPC only computes the output of a function – some additional tools are needed to enable the MPC computation to interact with the blockchain (see below).

[1] For example, as in the famous Beet Farmer auction [2]

Based on the above strengths and weaknesses, MPC is often either (1) used **within** a smart-contract [4]; or (2) along with time commitments, escrow accounts [1] and/or "sharded" digital signatures. These tools enable MPC to interface with the underlying blockchain. Some example applications of MPC in blockchains include:

- Lotteries [1]
- Decentralized Exchange (using private submitted bid and ask prices) [4]
- Collateral auctions (a Dutch auction of lending debts) [4]

4 Solving MPC

How can we solve the two millionaires problem? Consider a simple variant where Alice and Bob both have two bits of input, i.e. these are the two most significant bits of their input net worth value. So Alice's secret inputs are s_0, s_1 and Bob's secret inputs are t_0, t_1. First, we can write a boolean formula that takes these s and t values as inputs and outputs $TRUE$ iff Alice is at least as rich as Bob. Then, we can convert this boolean formula into an arithmetic formula using just $+$ and $*$ over $\mathbb{Z}_2$.

First, a boolean formula is:

$$F(s_1, s_0, t_1, t_0) = (s_1 \wedge \neg t_1) \vee [((s_1 \wedge t_1) \vee (\neg s_1 \wedge \neg t_1)) \wedge (s_0 \vee \neg t_0)]$$

Next, we can convert this to an equivalent arithmetic expression in $\mathbb{Z}_2$ using the following rules:

- $x \wedge y$ equals xy (i.e. x times y
- $x \vee y$ equals $x + y - xy$
- $\neg x$ equals $1 - x$

The above equations also work in any field like $\mathbb{Z}_p$ for p any prime, provided that the inputs are always either 0 or 1.[2] This will be important later when we want roots of unity (which exist in certain finite fields), in order to use the FFT.

When we do this with the above boolean formula, we get (from Gemini, not checked by me):

$$F(s_1, s_0, t_1, t_0) = s_1t_1 + s_1s_0t_0 + s_1t_0 + t_1s_0t_0 + t_1t_0 + t_1 + s_0t_0 + t_0 + 1$$

So then Alice and Bob want to compute the output of this arithmetic expression, which consists of multiple sums and multiplications, without revealing any information about s or t. How do they do it? First, they will compute shares of their inputs. Then they will perform operations over these shares. If they can compute both sums and products over *shares* then they can compute any function. As we'll see below, computing sums will be easy. Computing products will be the hard part.

Keep in mind that we'd eventually like to do this with an arbitrary number of players.

[2] What if we want to convert any non-zero input to 1 and keep the input 0 at 0? Hint: Fermat's Little Theorem can help!

5 Computing $\langle s+t \rangle$ via Shamir Secret Sharing

In Class Exercise. Say that there is a secret $s = 3$ and a secret $t = 5$ that is shared among Alice, Bob and Carol using Shamir sharing with the following functions:

Let

$$s(x) = 2x^2 + x + 3$$

and

$$t(x) = x^2 + 4x + 5$$

Recall that the secret is the y-intercept of the functions, so $s = 3$ and $t = 5$. We say that A, B and C hold the shares $\langle s \rangle$ and also the shares $\langle t \rangle$.

In particular, the three shares of each at the points -1, 1, 2 are

$s(1) = 6$, $s(-1) = 4$, $s(2) = 13$
$t(1) = 10$, $t(-1) = 2$, $t(2) = 17$

Now what if A, B, C add up their individual shares? Then, Alice, Bob and Carol will locally compute: $(s+t)(1) = 16$, $(s+t)(-1) = 6$ and $(s+t)(2) = 30$.

This gives them Shamir shares of the secret $s + t$.

So, Alice, Bob and Carol can get $\langle s+t \rangle$ from adding $\langle s \rangle$ and $\langle t \rangle$. Critically, they can do this **without revealing any info about the secrets s and t**

5.1 Some Questions

Here are some interesting questions about what Alice, Bob and Carol can compute shares of without revealing the secrets s and t.

1. Can they get $\langle 10(s+t) \rangle$? If so, How? Hint: Think in terms of changing **functions**.
2. Can they get $\langle 1 - 10(s+t) \rangle$? If so, How? Hint: Think in terms of changing **functions**.
3. Can they get $\langle st \rangle$?!? Does it work if they each just multiply $s_i * t_i$?

5.2 Additive Shares vs Shamir shares

OK so Shamir shares don't make it easy to get shares of $\langle st \rangle$. Here's another approach. What if we use *additive shares*? Additive shares of a secret s are simply shares s_i over the n players such that $\sum_i s_i = s$ in the field. Additive shares are simpler than Shamir shares, and are sometimes useful when Shamir shares are not. Later, we'll be seeing how we can go back and forth between these two types of shares efficiently using the Fast Fourier Transform (FFT).

But first, can we directly use additive shares to multiply without leaking information? If $s = \sum_i s_i$ and $t = \sum_i t_i$, then $st = (\sum_i s_i)(\sum_i t_i) = \sum_{i,j} s_i t_i$. So this doesn't look good. It seems that we'd need to have all players i and j exchange information in order to get all of these $s_i t_j$ shares. So this fails.

But, as we'll see below, there *is* a way we can use additive shares to multiply *if* we also make use of what are called Beaver triples.

5.3 Getting $\langle st \rangle$ using Beaver Triples

Imagine that our n players are given some addition shares to help them. In particular, they are given $\langle a \rangle$, $\langle b \rangle$ and $\langle ab \rangle$ where a and b are random numbers. These 3 additional shares are called a Beaver triple.

How do the players get them? Could be that they pre-compute a bunch of them every morning before coffee. Could be somebody else supples them to the players "as a service". We'll talk later about a way they can get them efficiently themselves on the fly by using an FFT-based algorithm.

But for now, let's see how they could use them if they had them. First, let

- $\alpha = s - a$
- $\beta = t - b$

Note that our friends can reveal α and β without revealing anything about s and t. Why? Because a and b are *random*! So let's have them do that so they all learn α and β. Then for each player i, they can compute the following share

- $u_i \leftarrow \beta s_i + \alpha b_i + c_i$

Then, we have

$$\begin{aligned}
\sum_i u_i &= \sum_i (\beta s_i + \alpha b_i + c_i) \\
&= \sum_i \beta s_i + \sum_i \alpha b_i + \sum_i c_i \\
&= \beta \sum_i s_i + \alpha \sum_i b_i + \sum_i c_i \\
&= \beta s + \alpha b + ab \\
&= (t - b)s + (s - a)b + ab \\
&= st
\end{aligned}$$

So the set of all the u_i then exactly equals $\langle st \rangle$. Moreover, they can be obtained without revealing anything about the secrets s and t!

Critically, the above Beaver triple trick can also be used when all the secrets are shared via Shamir shares. Just replace the sums in the above equations with interpolation of the polynomials.

This is important, because as we discussed in class Shamir shares are more robust than additive shares. In particular, if we create n Shamir shares of a $k < n$ degree polynomial, there is built-in redundancy. So the final secret can be reconstructed even if $n - k$ of the players drop out. Or for $k < n/2$, we can even handle $< n/2$ Byzantine players who actively send out incorrect information.

6 How do we get Beaver Triples?

For decentralized finance, we'd like to be able to perform MPC when the number of players n is very large: think "millions of millionaires". In particular, we'd like the latency and the number of message sent per player to scale sub-linearly with n. In fact, it's better if these costs can be just polylogarithmic in n.

How can we achieve this? Basically, we can use the FFT! TODO.

References

[1] Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Łukasz Mazurek. Secure multiparty computations on bitcoin. *Communications of the ACM*, 59(4):76–84, 2016.

[2] Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, et al. Secure multiparty computation goes live. In *International Conference on Financial Cryptography and Data Security*, pages 325–343. Springer, 2009.

[3] Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In *2020 IEEE symposium on security and privacy (SP)*, pages 910–927. IEEE, 2020.

[4] Yunqi Li, Kyle Soska, Zhen Huang, Sylvain Bellemare, Mikerah Quintyne-Collins, Lun Wang, Xiaoyuan Liu, Dawn Song, and Andrew Miller. Ratel: Mpc-extensions for smart contracts. In *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security*, pages 336–352, 2024.

[5] Loi Luu, Jason Teutsch, Raghav Kulkarni, and Prateek Saxena. Demystifying incentives in the consensus computer. In *Proceedings of the 22Nd acm sigsac conference on computer and communications security*, pages 706–719, 2015.

[6] Kaihua Qin, Liyi Zhou, and Arthur Gervais. Quantifying blockchain extractable value: How dark is the forest? In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 198–214. IEEE, 2022.

[7] Daria Smuseva, Andrea Marin, Sabina Rossi, and Aad Van Moorsel. Verifier's dilemma in proof-of-work public blockchains: A quantitative analysis. *ACM Transactions on Modeling and Computer Simulation*, 35(2):1–24, 2025.