Improving HIV treatment choice with multi-party computation

Mark Abspoel

6 April 2018

Based on TKI project with TNO, CWI, UvA/IAS, Philips.
Joint work with: Thomas Attema (CWI/TNO), Ronald Cramer (CWI), Serge Fehr (CWI), Jan de Gier (TNO), Maran van Heesch (TNO), Pia Kempker (TNO), Emiliano Mancini (UvA/IAS), Gabriele Spini (CWI), Thijs Veugen (CWI/TNO), Daniël Worm (TNO).

Choosing HIV treatment

- Treating HIV is not straight-forward: multiple possible treatments, many different viruses
- Virus mutates as it replicates. Bad treatment leads to more replication, which means:
  - Treatment failure
  - Accumulation of drug resistances
  - Faster progression to AIDS
- Even with optimal treatment, virus will eventually mutate

Doctors' decisions

Doctors take decisions based on

- Yearly published guidelines based on current research
- Knowledge
- Experience

Every time a patient needs new treatment, they get feedback on the results of the old treatment.

Can we use this data?

Statistics on treatment decisions and results

Two problems:

1. Doctors do not want to publish decisions for liability concerns
2. Even if there was a database somewhere, patient's HIV genotype is privacy-sensitive

Statistics on treatment decisions and results

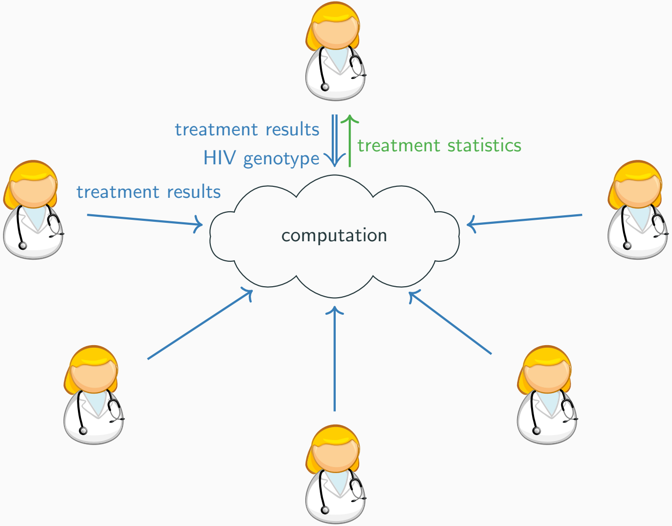
Two problems:

1. Doctors do not want to publish decisions for liability concerns
2. Even if there was a database somewhere, patient's HIV genotype is privacy-sensitive

Solution: multi-party computation!

Multi-party computation

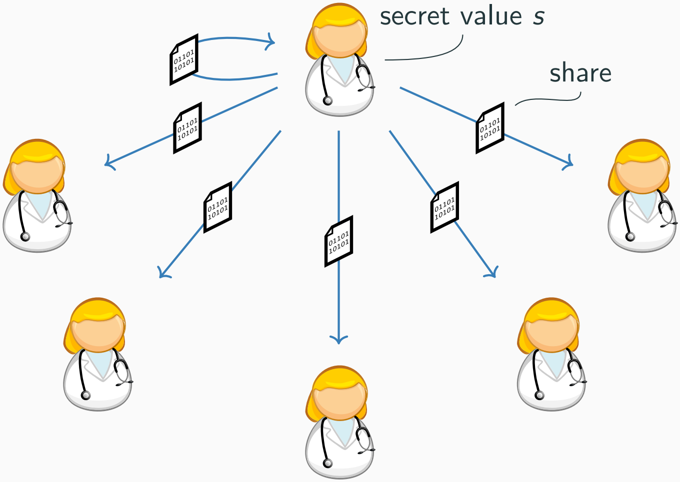
Given a patient's HIV genotype, per treatment: what was the average time to failure for patients with similar HIV virus?



Multi-party computation

Secret sharing

To compute on other parties' data, we need a way to distribute secret data. → **Secret sharing**

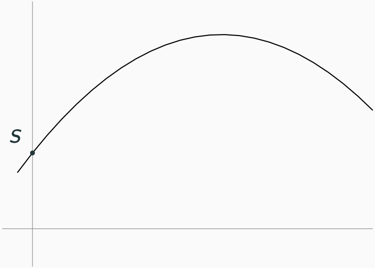


Shamir's secret sharing scheme

Suppose we have a secret value s in some finite field $\mathbb{F}$, and we want to disperse it into n shares.

Fix an integer $0 < t < n$, the *threshold*.

Pick a uniformly random secret polynomial $f = s + c_1 X + c_2 X^2 + \cdots + c_t X^t$ with $c_1, \ldots, c_t \in \mathbb{F}$.

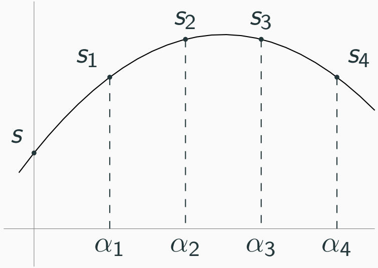


Shamir's secret sharing scheme

Suppose we have a secret value s in some finite field $\mathbb{F}$, and we want to disperse it into n shares.

Fix an integer $0 < t < n$, the *threshold*.

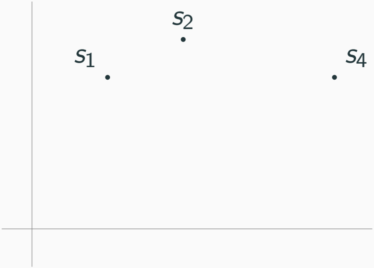
Pick a uniformly random secret polynomial $f = s + c_1X + c_2X^2 + \cdots + c_tX^t$ with $c_1, \ldots, c_t \in \mathbb{F}$.



$s_i := f(\alpha_i)$ is the i-th share.

Theorem (Lagrange interpolation)

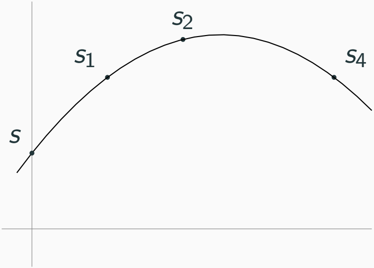
Given $y_1, \ldots, y_{t+1} \in \mathbb{F}$ *there is a unique polynomial* $h = a_0 + a_1 X + \cdots + a_t X^t$ *with* $h(\alpha_i) = y_i$ *for* $i = 1, \ldots, t+1$.



Reconstructing the secret

Theorem (Lagrange interpolation)

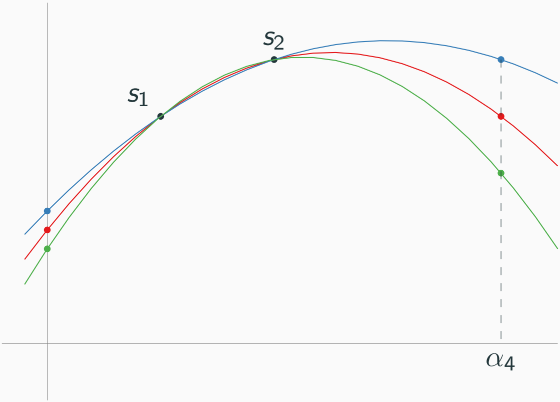
Given $y_1, \ldots, y_{t+1} \in \mathbb{F}$ there is a unique polynomial $h = a_0 + a_1 X + \cdots + a_t X^t$ with $h(\alpha_i) = y_i$ for $i = 1, \ldots, t+1$.



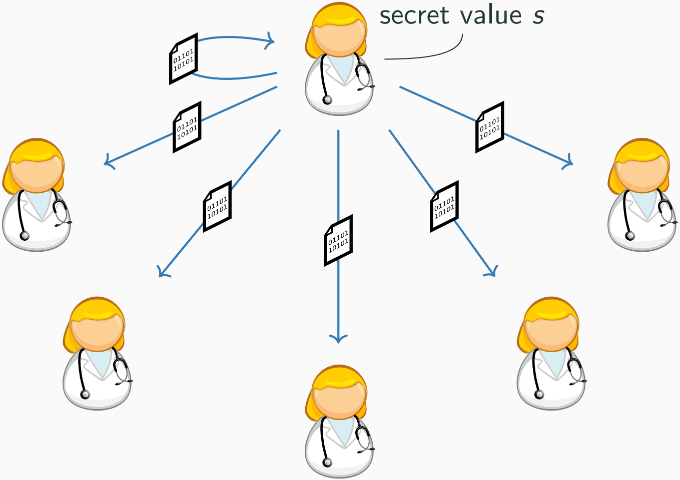
So: given $t + 1$ shares we can reconstruct f.

Privacy

Given $\leq t$ shares, we get no information about the secret s.

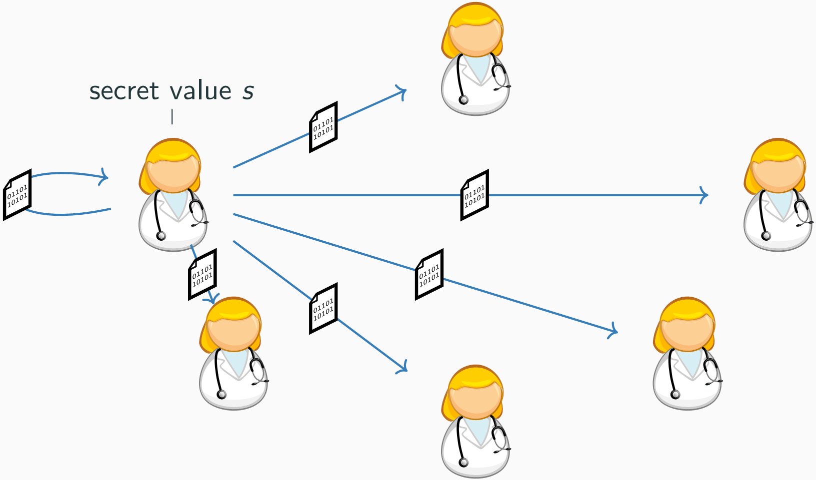


MPC: Step one



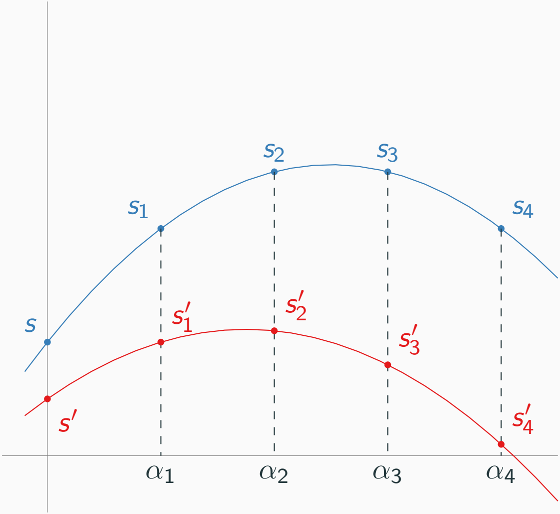
Every party shares their secret inputs to the computation. Each party has one share of every secret value.

MPC: Step one

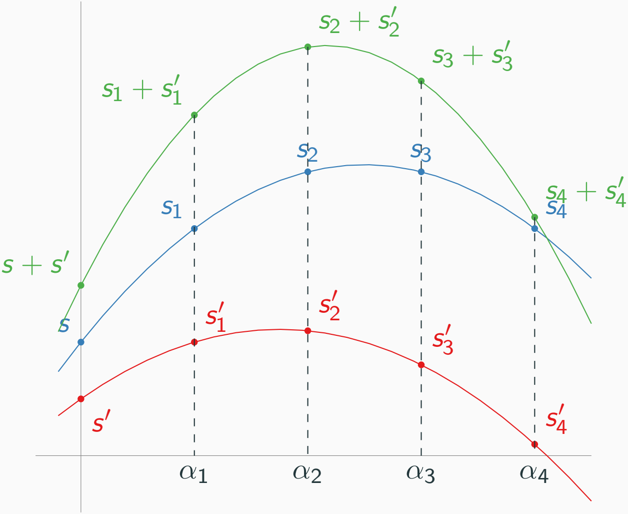


Every party shares their secret inputs to the computation. Each party has one share of every secret value.

Adding shares



Adding shares



Each party can calculate a share of $s + s'$.

Linear functions

For $c \in \mathbb{F}$ a publicly known scalar, parties can also calculate a share of $c \cdot s$ by multiplying their own share.

So, now we can evaluate linear forms $\mathbb{F}^n \to \mathbb{F}$, e.g.

$$(x_1, \ldots, x_n) \mapsto a_1 x_1 + \cdots + a_n x_n$$

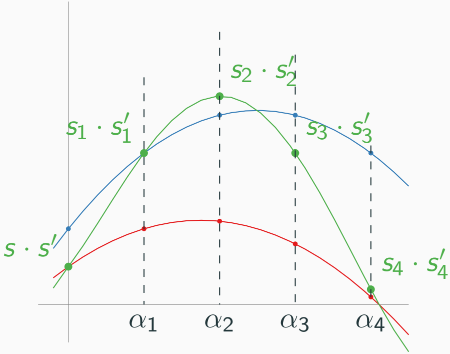
for constants $a_1, \ldots, a_n \in \mathbb{F}$.

What about arbitrary functions $\mathbb{F}^n \to \mathbb{F}$? We need multiplication.

Multiplication

Suppose we have secrets s, s' corresponding to polynomials f, g.

$\deg fg = (\deg f) \cdot (\deg g)$. Hence we need $2t + 1$ shares to reconstruct ss'.



We can only do this a limited number of times before we run out of shares.

Multiplication

Suppose we have secrets s, s' corresponding to polynomials f, g.

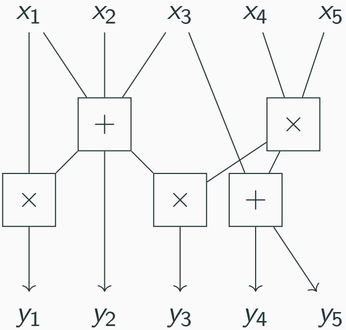
Set $x_i := s_i \cdot s'_i$. Party i can calculate this value. Now they secret-share x_i with each other party. Write $[x_i]_j$ for the share of party j of the value x_i.

Note that the function mapping $L : (s_1, s_2, \ldots, s_n) \mapsto s$ is a linear form!

So every party can calculate $L([x_1]_i, \ldots, [x_n]_i) = [s \cdot s]_i$.

Evaluating arbitrary functions

We can represent any computable function $\mathbb{F}^m \to \mathbb{F}^k$ as a circuit of linear gates and multiplication gates.



E.g. a Boolean AND-gate is just multiplication $b \cdot b'$ if inputs are b, b' encoded as $\{0, 1\}$ in $\mathbb{F}$.

In general: circuit can be very big!

Multi-party computation

Assuming we have the desired function to be computed f as a circuit:

1. Every party secret-shares their input
2. Parties locally process linear gates, and communicate for every multiplication gate
3. Every party sends their shares of the output values to the parties who should receive the output, who then reconstruct the value.

Note: we assume parties strictly follow the protocol.

Summary of MPC

We have seen the basics of how MPC works.

Linear operations: cheap. Multiplications require interaction, which is the dominating cost.

Big circuits $\rightarrow$ slow computation.

HIV treatment

Given a patient's HIV genotype, per treatment: what was the average time to failure for patients with similar HIV virus?

- Every doctor has a list of ($\texttt{HIV genotype}, \texttt{time to failure}$). We need to secret share each of those values.
- HIV genotype gets encoded as a vector $\mathbf{m} \in \mathbb{F}^{\ell}$.
- Want to compute
$$\frac{\sum_{i=1}^{N} 1_{\mathbf{m}_i \approx \mathbf{m}} t_i}{\sum_{i=1}^{N} 1_{\mathbf{m}_i \approx \mathbf{m}}}$$
- Need to check against *every* prior patient: computation scales linearly in ℓ, N

Doctor has 5 minutes per patient to make a treatment decision. Work-in-progress: can we make computation fast enough?

Some research topics

- Can we do MPC over finite rings instead of fields, e.g. $\mathbb{Z}/2^m\mathbb{Z}$?
- How can we do often used functionalities efficiently, e.g. integer division?
- Can we do machine learning tasks over MPC, e.g. decision trees and random forests?