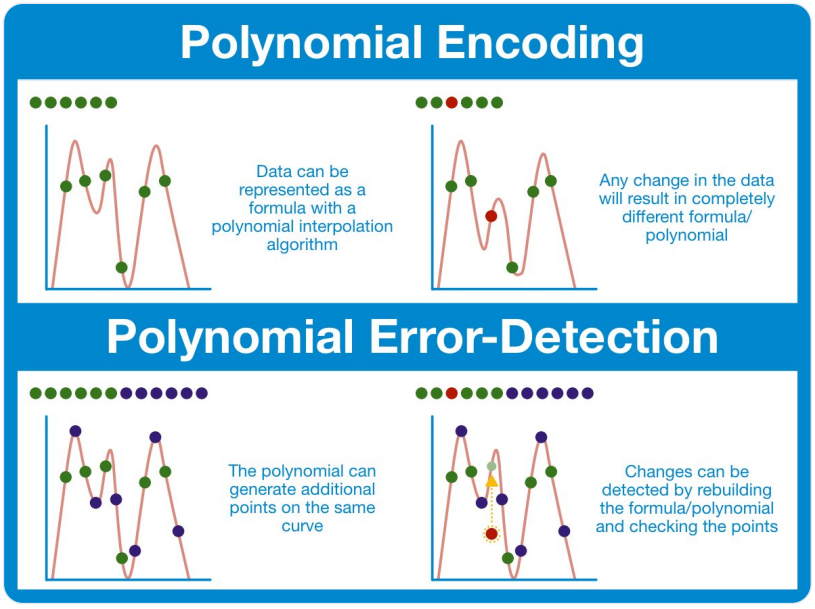
Haym @SalomonCrypto

Oct 23 • 19 tweets • SalomonCrypto/status/1584212843938734081

(1/19) Cryptography Basics: Polynomial Erasure Codes

Moving data across the internet is difficult; assuming you trust all the intermediate parties, you still have vicious network conditions. How can you ensure your data stays clean?

A lesson in pratical cryptography.

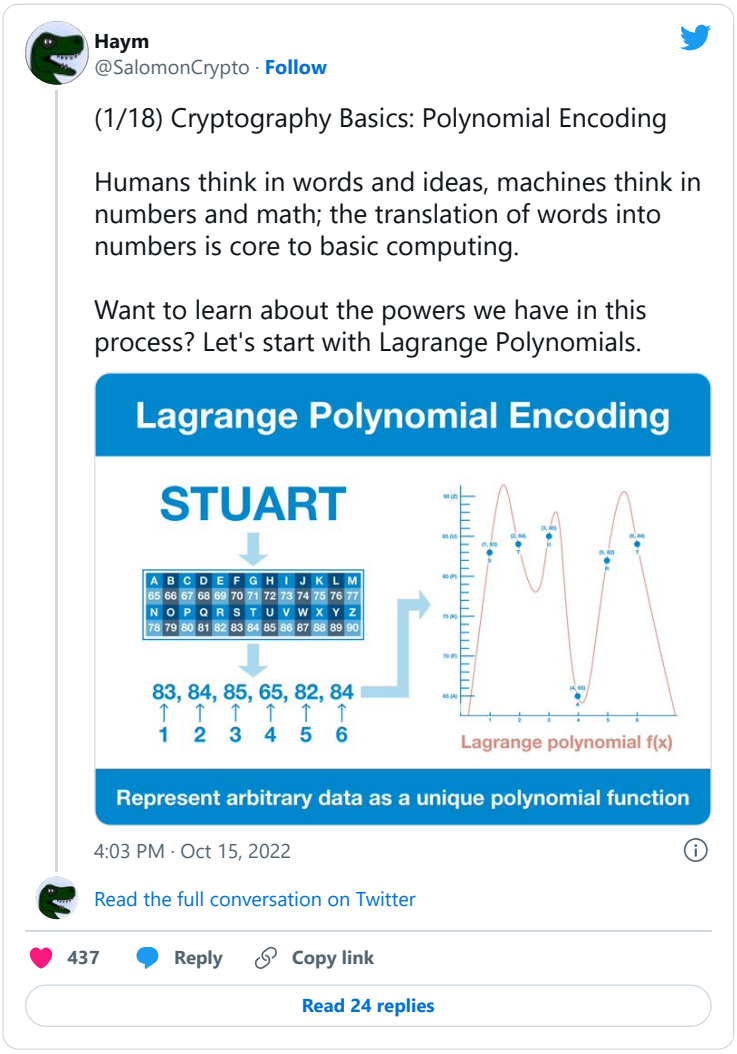


(2/19) Encoding is the process of mapping one set of symbols (usually human readable, like letters/words) to another (usually machine readable, like numbers/math).

One purpose of encoding is to translate data, another is to provide additional functionality.

(3/19) Polynomial encoding is a specific type of encoding that packages arbitrary data into a mathematical function know as a polynomial.

Tl;dr the dataset and the polynomial convey the same underlying data, albeit expressed as a function (as opposed to a list of data points)



(4/19) Polynomial encoding is incredibly powerful, we've already seen how they can underpin a cryptographic commitment scheme.

But that's just the tip of the iceberg. Polynomials give us so much more functionality than binding datasets.

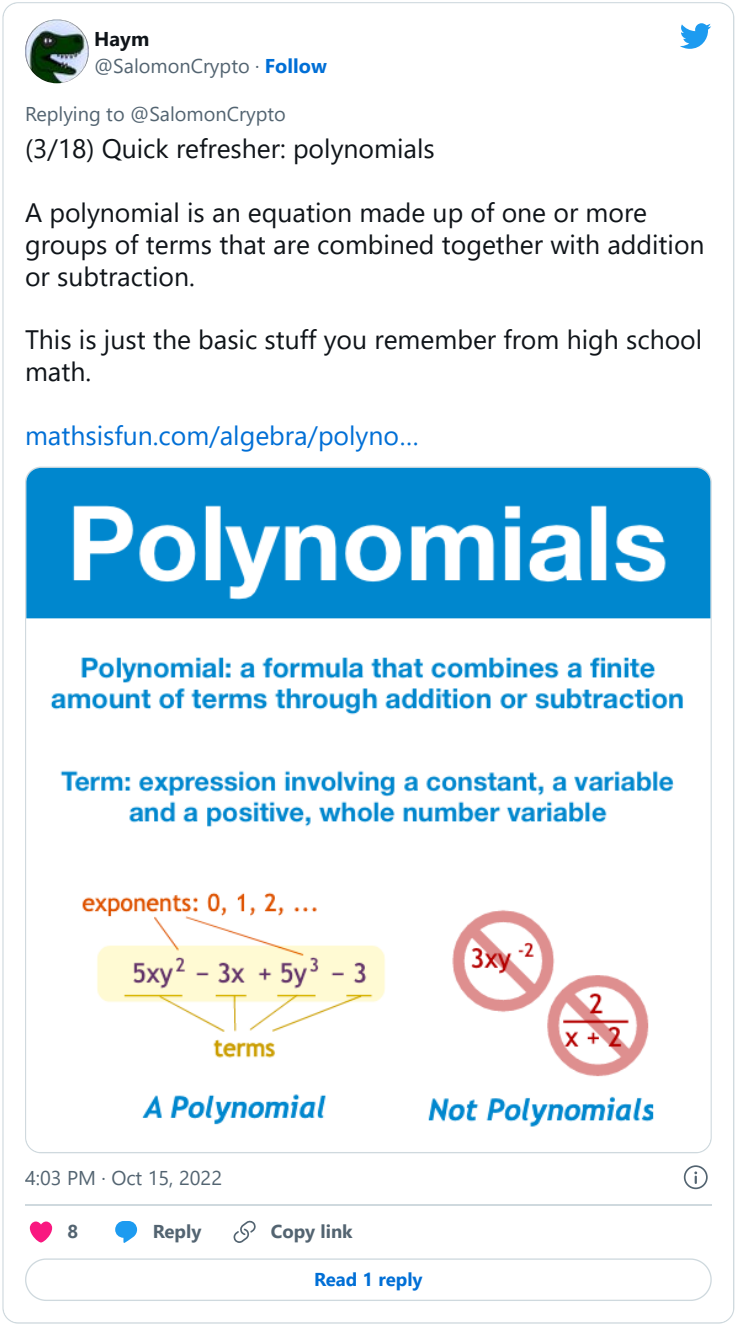
(5/19) If you've been paying close enough attention, you may have noticed a pattern: when we have n data points, our polynomial always seems to have at most n terms (or maybe you've notice that the highest exponent is always n - 1).

Let's expand on that.

(6/19) A polynomial is an expression made up of the addition/subtraction of terms. Terms are made up of a variable, raised to some power, times a constant (coefficient).

The degree of the polynomial is the highest exponent.

Degree of x^2+x-5 is 2
Degree of 4x^5-6x^2+6 is 5



(7/19) So rewind back to polynomial encoding. It turns out that in order to generate a line that passes through n data points, the associated polynomial will have (at most) degree n-1.

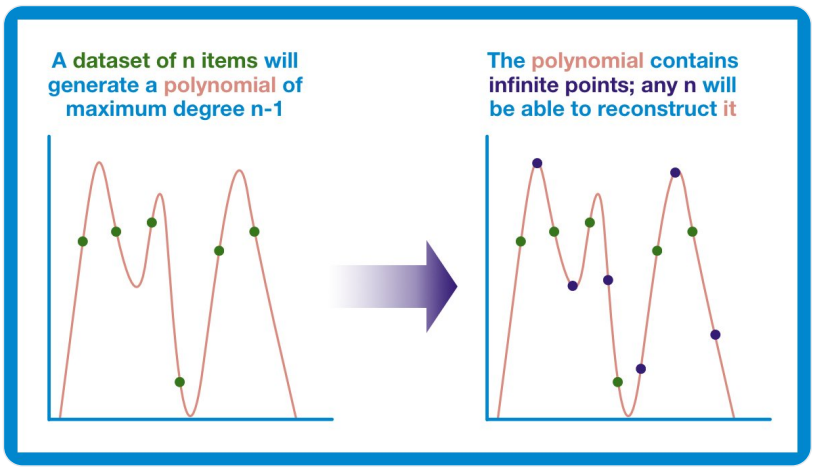
Encoding 10 datapoints? We're (probably) going to need a polynomial that has a term with x^9.

(8/19) We can also use this principle in reverse: to recreate a polynomial of degree n-1, we need (at least) n points that fall on that line.

And remember, a polynomial is a formula that can generate an infinite amount of outputs - part of the original dataset or brand new.

(9/19) If our original dataset is n pieces, we need a polynomial of degree n-1... but we can grab more points from that same line.

Now, we have more than n points; if one or two get lost or corrupted, we still have more than enough points to reconstruct our polynomial.

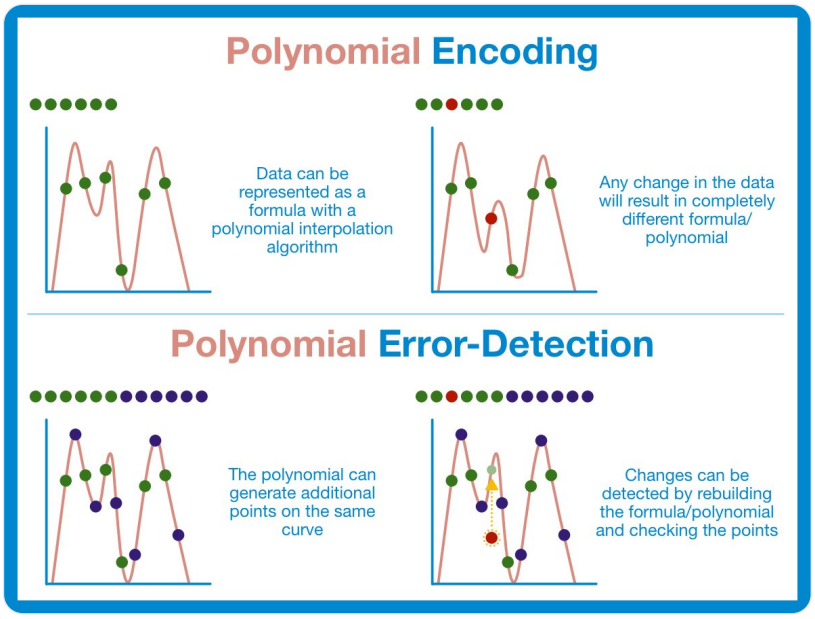


(10/19) This is the key insight we are going to build an error-correcting code from.

Error-correcting codes are a family of encoding that add some additional information to a dataset that allows the recovery of the original data, even if some of the pieces were lost.

(11/19) After we create our data polynomial, we are going to use it to generate additional points along the same line.

These additional points convey enough extra information that we can reconstruct the underlying polynomial and use it to double check all the points.



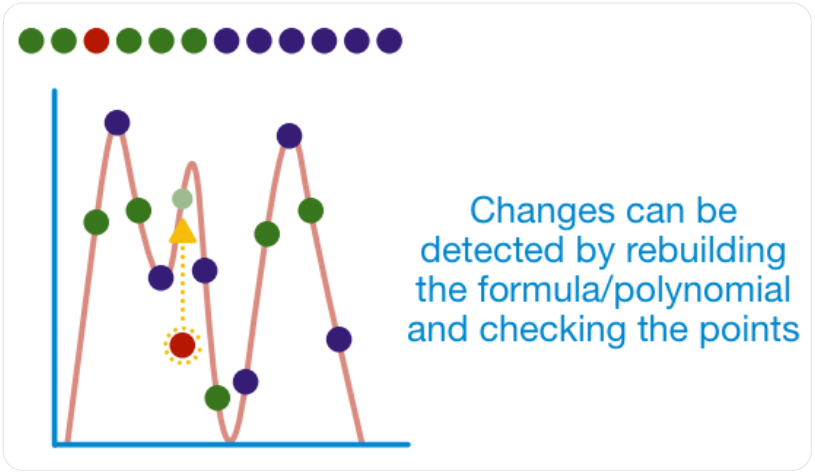
(12/19) Recipe for a polynomial error-correcting scheme:

Take the original data set of length n and generate a polynomial. Use the polynomial to find k extra points. Append the extra points on to the original data.

We can now detect up to k errors in our code.

(13/19) In fact, polynomial error-detection is so powerful that we can use it for more than just detection. Polynomial encoding can be used for data correction!

All we need to do is figure out which points fall off the line and move them to where they need to be.



(14/19) At this point, you understand the important parts:

- any arbitrary data can be plotted
- you can generate a unique polynomial for that data
- this polynomial can find extra points
- these extra points are used to confirm the original data

(15/19) There's one more application of polynomial error-schemes: erasure codes.

We begin with all the same steps as before: build the polynomial, generate the extra data. The only difference is that instead of looking for errors, we are just interested in recovery.

(16/19) Let's take a look at Reed-Solomon codes. This encoding scheme can be deployed for error detection, correction, or as an erasure code.

When deployed as an erasure code, for every k pieces of additional data, it can recover up to k lost pieces.

https://en.wikipedia.org/wiki/Reed%E2%80%93Solomon_error_correction

(17/19) Fun fact, almost all two-dimensional bar codes (including QR codes) use Reed-Solomon encoding.

That's right, the same technology that powers QR codes is (spoiler alert) at the very core of Danksharding.

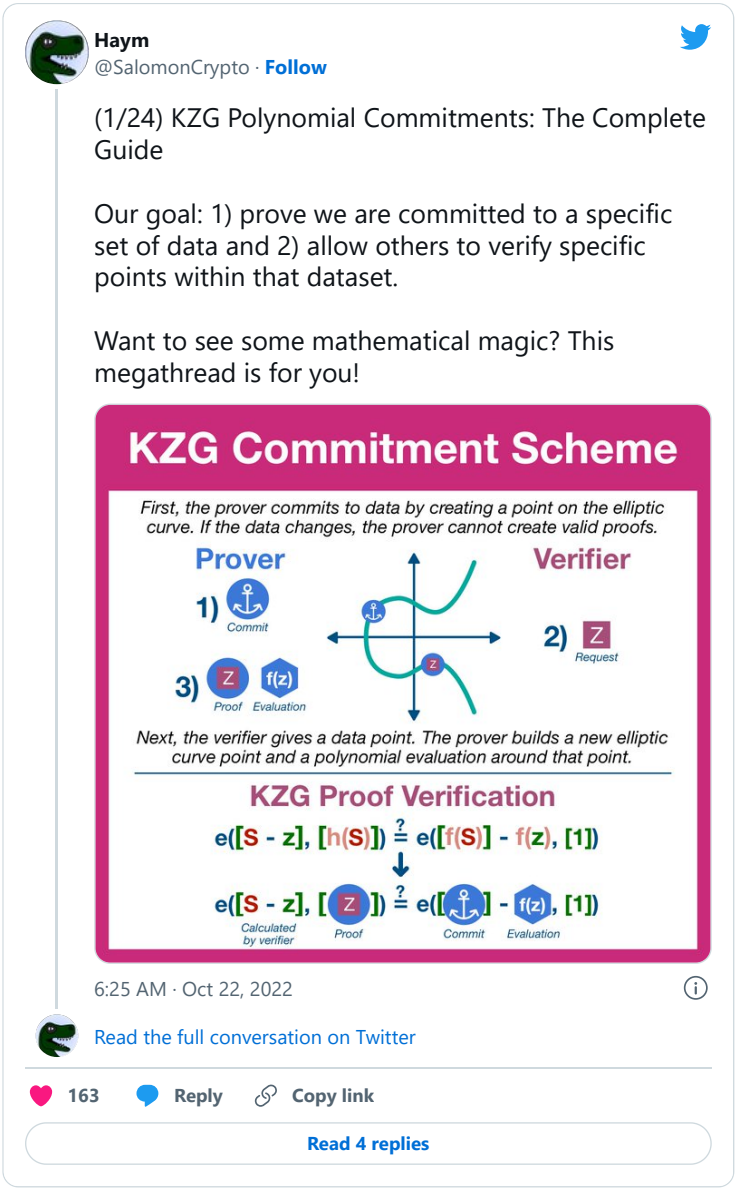
It's math the COOLEST?!?!

(18/19) Remember, this all falls back on the principle we discussed earlier: to recreate a polynomial of degree n-1, we need (at least) n points that fall on that line.

The extra points provided by the Reed-Solomon code make sure there are always more than n available points.

(19/19) Alright anon, you've got it: polynomial erasure coding transforms data into a polynomial and adds more data to ensure the data is recoverable from just a subset.

Now... what else can we do with a polynomial...



• • •