Shamir secret sharing

Tanja Lange

Eindhoven University of Technology

2WF80: Introduction to Cryptology

Motivation

In the encryption / signature / KEM systems we have seen, the private key has a lot of power.

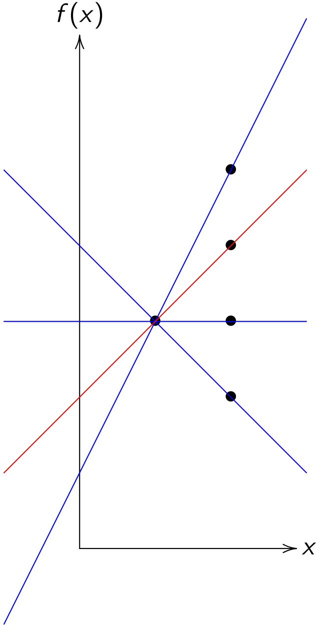
Many structures are set up so that multiple people must contribute to perform an action – think of opening a bank vault with physical keys.

We deal with the simplest case, that all users are equal and that a certain number of them need to contribute, this is called a threshold system.

We share a secret among N users in a way that any t of them can recover it, while $t-1$ or fewer get no information on it.
This is called a t-out-of-N system.

Can emulate more powerful users by giving them more shares.

Idea

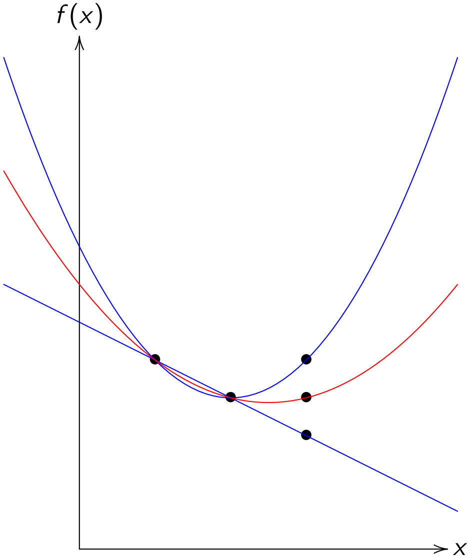


A line is uniquely determined by two points.

Knowing only one point holds no information about where the line intersects the y-axis:

Any of the blue lines is a candidate line.

Idea

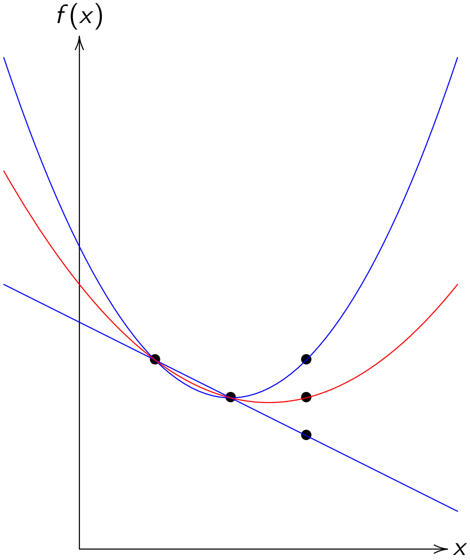


A degree-2 polynomial is uniquely determined by three points.

Knowing only two or fewer points holds no information about where the function intersects the y-axis:

Any of the blue graphs is a candidate.

Idea



A degree-2 polynomial is uniquely determined by three points.

Knowing only two or fewer points holds no information about where the function intersects the y-axis:

Any of the blue graphs is a candidate.

In general, t points uniquely define a polynomial of degree $\leq t-1$.

Shamir secret sharing

To share integer a do the following:

Generate polynomial:
Pick $t-1$ random integer coefficients $f_1, f_2, \ldots, f_{t-1}$ and define

$$f(x) = a + \sum_{i=1}^{t-1} f_i x^i.$$

(This polynomial satisfies $f(0) = a$.)

Generate shares:
Each user receives one secret share $(i, f(i))$;
Note that here $i \neq 0$ and $i \neq j$ must hold.
(This matches a point in the graph.)

Shamir secret sharing

To share integer a do the following:

Generate polynomial:
Pick $t-1$ random integer coefficients $f_1, f_2, \ldots, f_{t-1}$ and define

$$f(x) = a + \sum_{i=1}^{t-1} f_i x^i.$$

(This polynomial satisfies $f(0) = a$.)

Generate shares:
Each user receives one secret share $(i, f(i))$;
Note that here $i \neq 0$ and $i \neq j$ must hold.
(This matches a point in the graph.)

Then t users can reconstruct $f(x)$ by Lagrange interpolation,
while $t-1$ or fewer users learn nothing about $f(0)$.

Shamir secret sharing

To share integer a do the following:

Generate polynomial:
Pick $t-1$ random integer coefficients $f_1, f_2, \ldots, f_{t-1}$ and define

$$f(x) = a + \sum_{i=1}^{t-1} f_i x^i.$$

(This polynomial satisfies $f(0) = a$.)

Generate shares:
Each user receives one secret share $(i, f(i))$;
Note that here $i \neq 0$ and $i \neq j$ must hold.
(This matches a point in the graph.)

Then t users can reconstruct $f(x)$ by Lagrange interpolation,
while $t-1$ or fewer users learn nothing about $f(0)$.

Note that the shares $(i, f(i))$ are secret information and must be transmitted in an encrypted manner.

Lagrange interpolation

We can recover the entire polynomial $f(x)$ from t shares, but we only care about $f(0) = a$.

Let users with shares $i_1, i_2, \ldots, i_t$ with $i_j \neq i_k$ participate in the reconstruction. Then

$$f(0) = \sum_{j=1}^{t} f(i_j) \prod_{k=1, k\neq j}^{t} i_k/(i_k - i_j).$$

The product is over $t-1$ fractions for each summand. Excluding $k = j$ avoids division by zero.

If more than t users contribute, just ignore the surplus shares.

Security considerations

Nobody should ever know a.

Security considerations

Nobody should ever know a.

Once the t parties computed a they don't need the others anymore.

Security considerations

Nobody should ever know a.

Once the t parties computed a they don't need the others anymore.

This is solved (in theory) by using a trusted party who gets the shares, computes and uses a, and then forgets the shares and a.

Security considerations

Nobody should ever know a.

Once the t parties computed a they don't need the others anymore.

This is solved (in theory) by using a trusted party who gets the shares, computes and uses a, and then forgets the shares and a.

It's much better to use the shares locally to perform a partial decryption / signature and then to combine those parts using the Lagrange coefficients.
This uses additivity of the shares and depends on the scheme.
See exercise sheet 7 for more.

Security considerations

Nobody should ever know a.

Once the t parties computed a they don't need the others anymore.

This is solved (in theory) by using a trusted party who gets the shares, computes and uses a, and then forgets the shares and a.

It's much better to use the shares locally to perform a partial decryption / signature and then to combine those parts using the Lagrange coefficients.
This uses additivity of the shares and depends on the scheme.
See exercise sheet 7 for more.

We cannot trust anybody to forget secrets, so generate a in a distributed manner as well by having t users contribute.
Each of the t users then shares their input in a t-out-of-N manner.

A user should get all his shares for the same i so that he can combine the t shares into one.