**A Simple and Profound Understanding of MPC - The Cryptography behind OKX MPC Wallet**

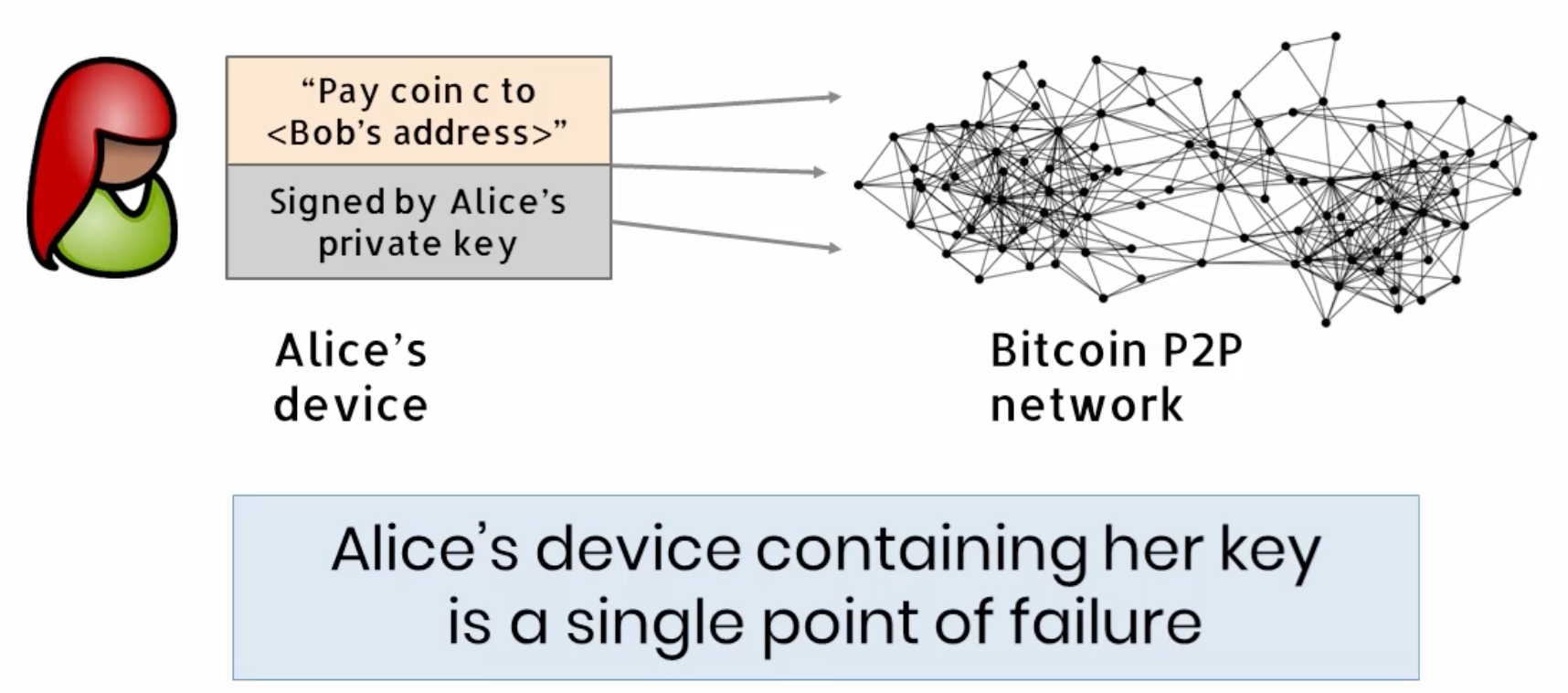
This session is an attempt to explain the cryptography behind the mpc wallet, so some mathematics are involved. However, in order to keep simple, the mathematical strictness is kind of ignored.

**High school level mathematics would be enough to understand most of it.**

1. **OKX MPC wallet**

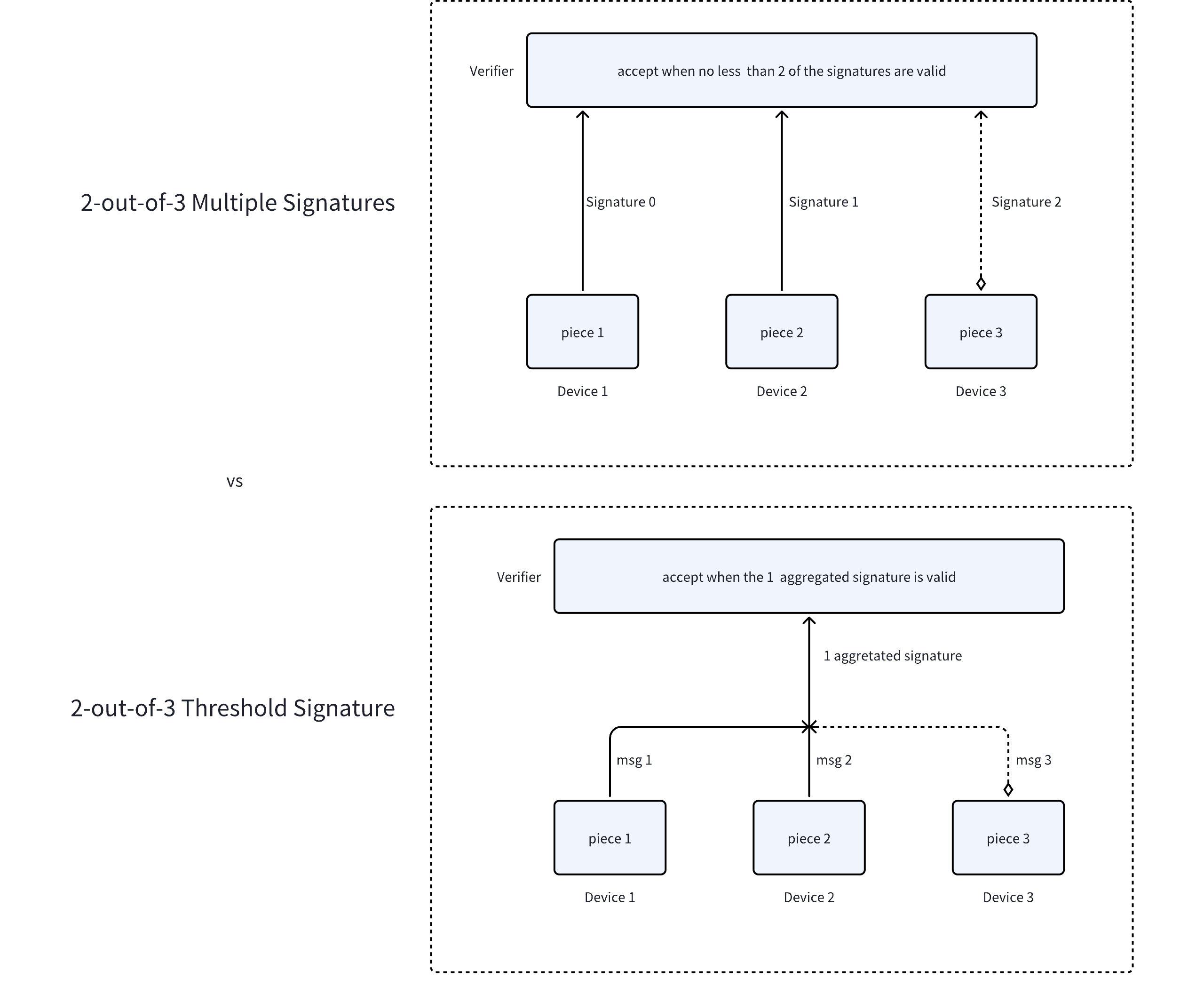
**1.1 Single Point Failure**

One of the most critical issues for users entering web3 is the "**single point failure**".



What if we lost our private keys? Or keys being stolen?

**1.2 Multiple signatures vs threshold signature**



**1.3 t-out-of-n threshold signature**

The secret key is split and shared among **n** players.

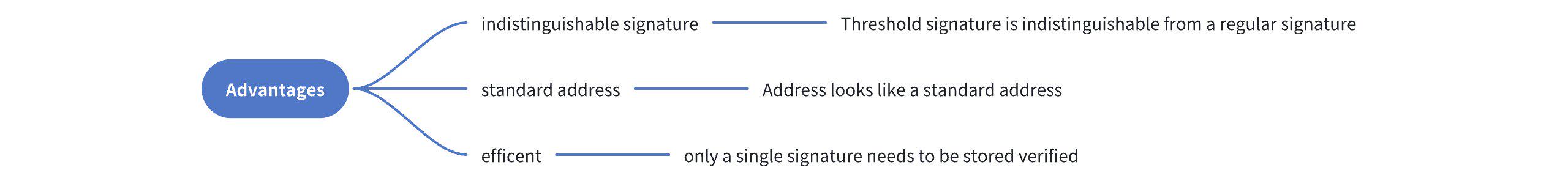
* Correctness

Any **t** of them can jointly sign any given message.

* Security

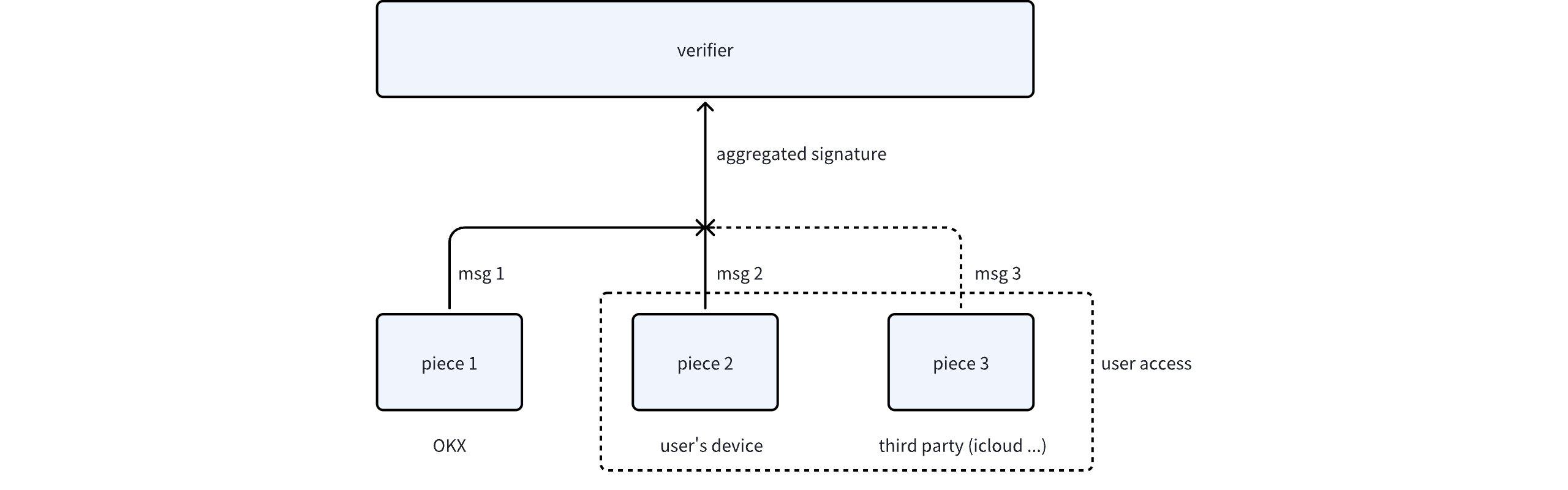
No **t-1** concluding players can forge any signature.

**1.4 Advantages of threshold signature**



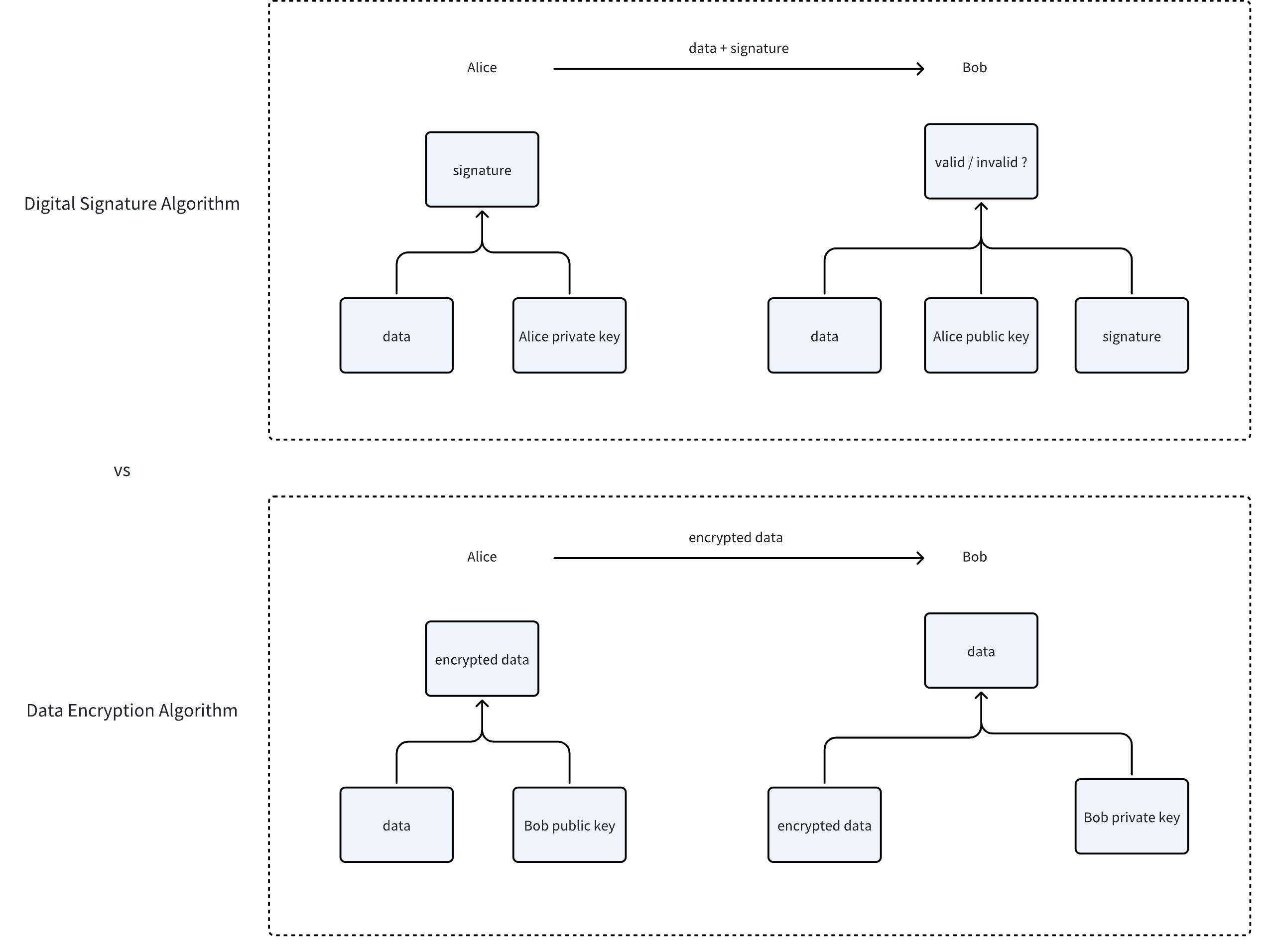
**1.5 OKX MPC wallet**

We developed a 2-out-of-3 threshold signature wallet based on Multi-Party Computation (MPC).

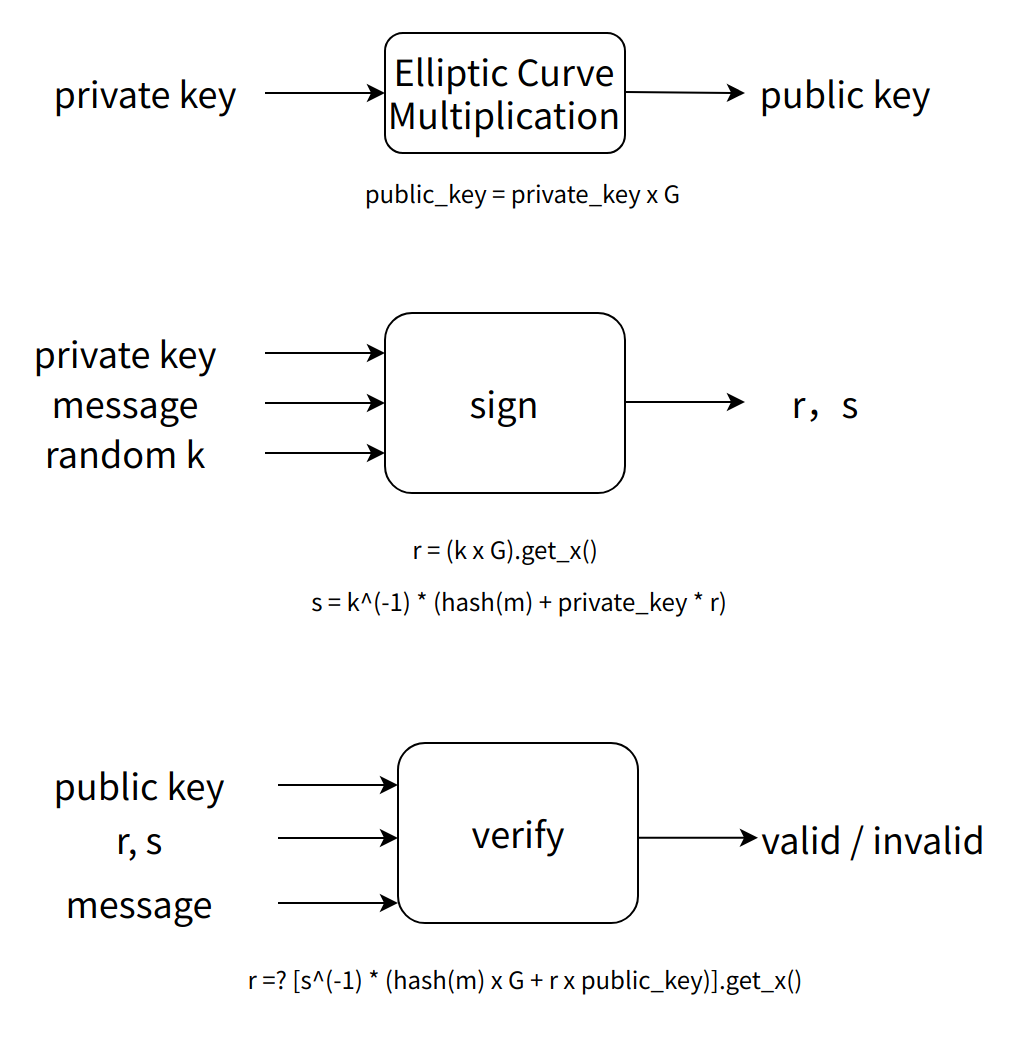


2. **Digital Signature Algorithm**

DSA is used to create a signature for any given message, so that we can **validate the authenticity of the message**.



**2.1 Elliptic Curve Digital Signature Algorithm (ECDSA)**

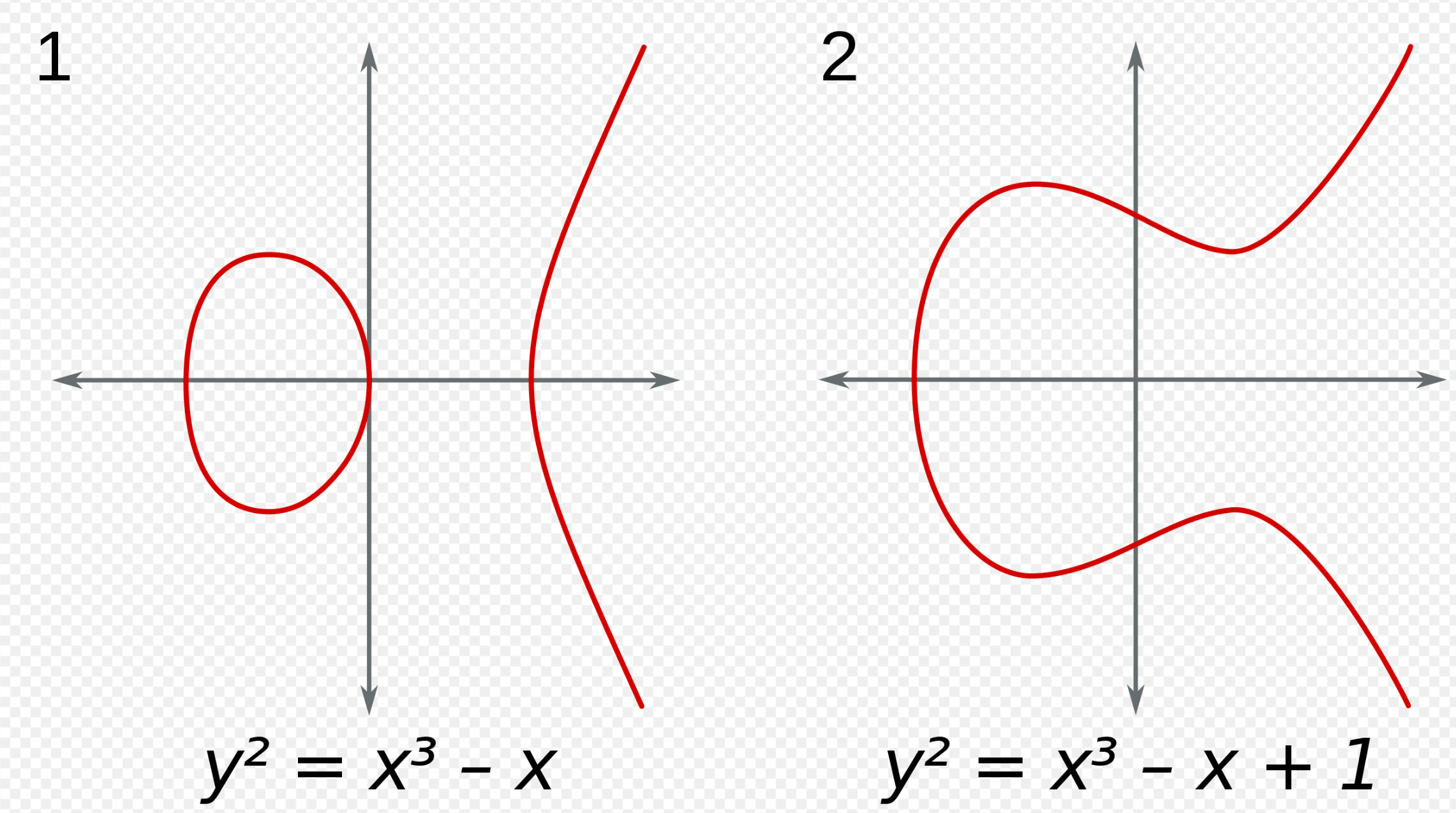


Be noted that the operation of "" is denoted as [elliptic curve point multiplication by a scalar](https://en.wikipedia.org/wiki/Elliptic_curve_point_multiplication).

G is a public parameter, the generator of the group.

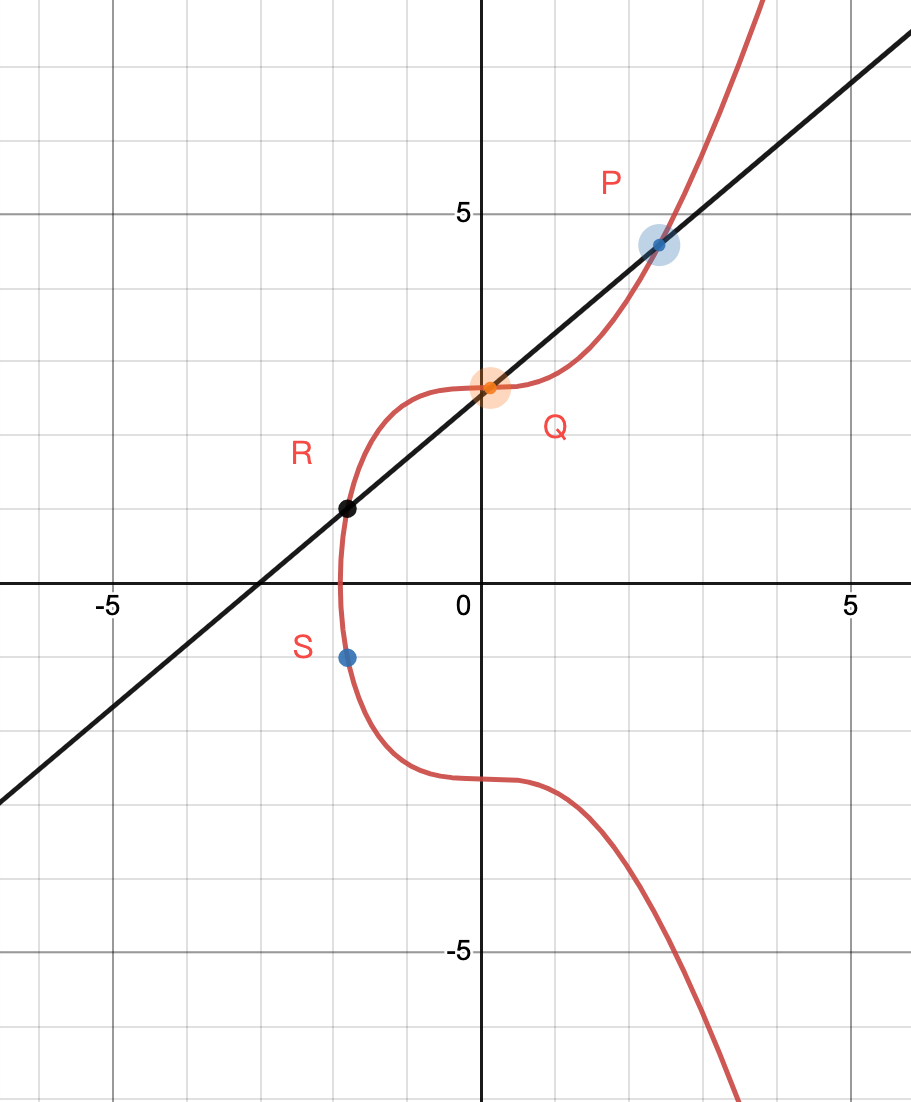
**2.2 Elliptic Curve Operation**

**2.2.1 Elliptic Curve Equation**



**2.2.2 Elliptic Curve Operations**

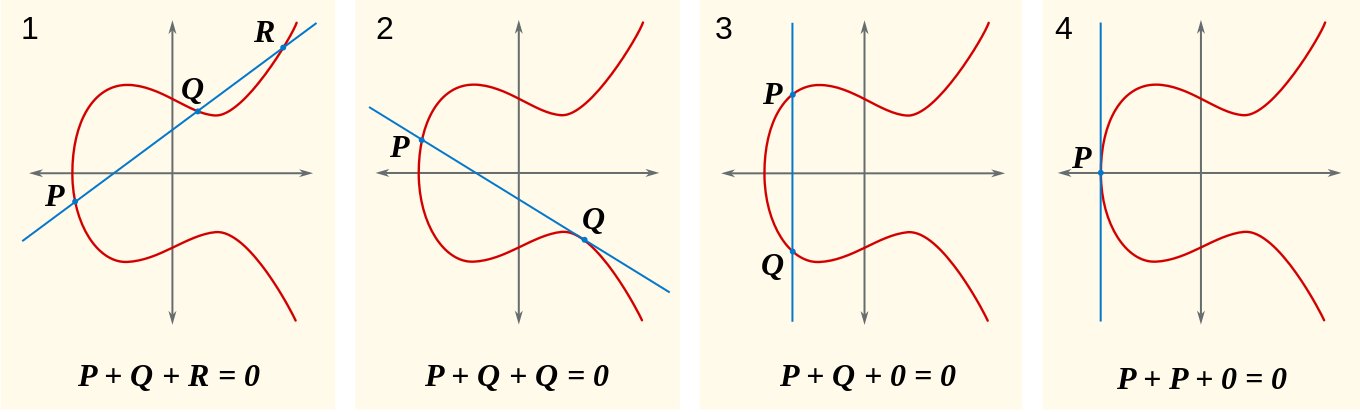
P + Q = S



https://www.desmos.com/calculator/ialhd71we3

Specifically, we define an infinite point 0，satisfying P + 0 = P.

Elliptic curve point operations: Addition (shown in facet 1), doubling (facets 2 and 4) and negation (facet 3).



Scalar Multiplication:

More details can be found at https://en.wikipedia.org/wiki/Elliptic\_curve

**2.3 Why is your private key safe?**

The private key is safe because the EC operation is defined in a finite field, in which all numbers need be moded by a specified prime.

For instance, in the finite field of prime , .

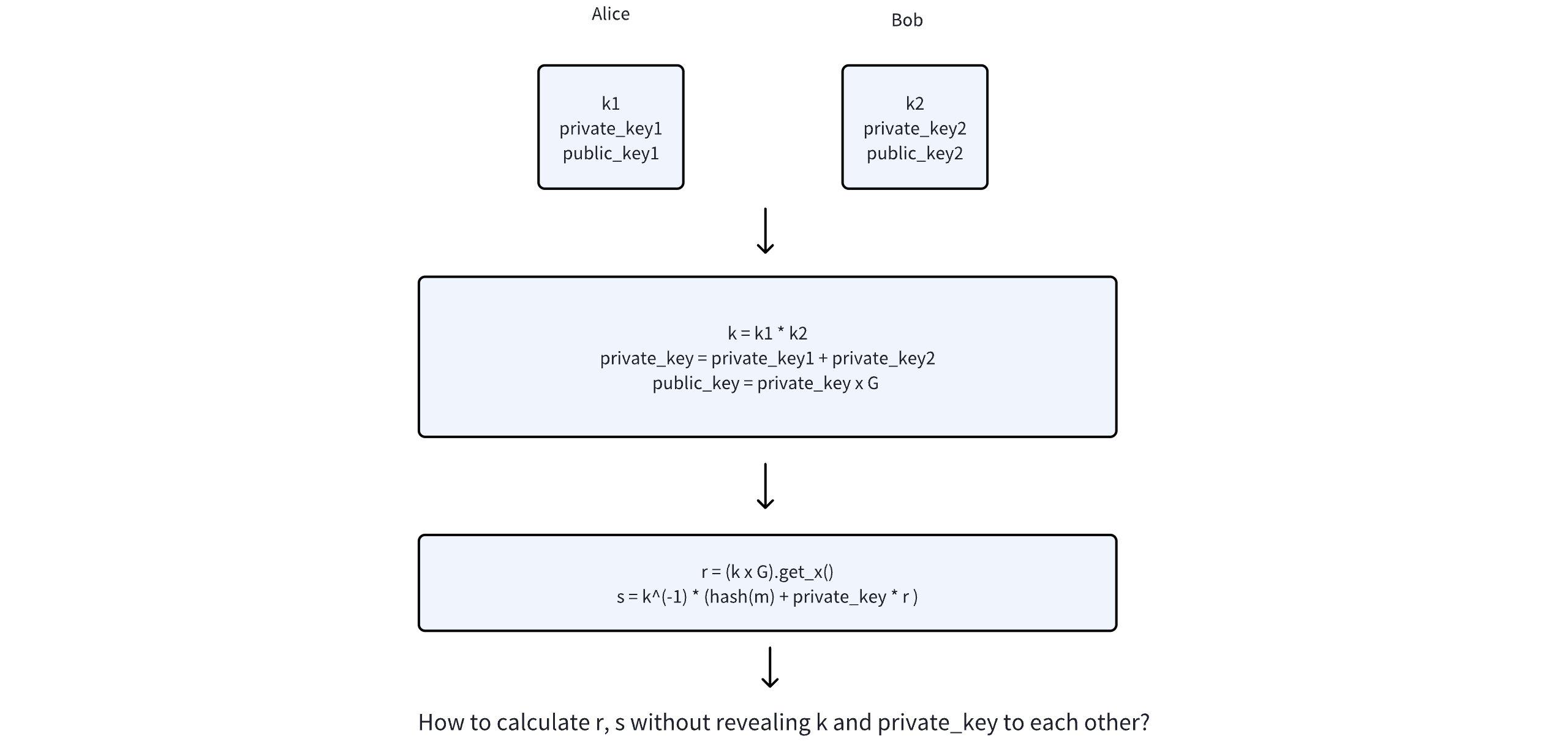
So there is no efficient way to calculate the private key other than calculating all points in this group. For a group with 256 bits order, there are roughly 2^256 values in total, so the reversing is not practical.

**2.4 Other Signature schemes**

There are quite a lot of other signature algorithms, such as Schnorr, BLS etc. though we focus on ECDSA here, especially the curve of **secp256k1**, which is widely used in Bitcoin and Ethereum.

|  |
| --- |
| Plain Text // secp256k1 p = FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE FFFFFC2F a = 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 b = 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000007 Gx = 79BE667E F9DCBBAC 55A06295 CE870B07 029BFCDB 2DCE28D9 59F2815B 16F81798 Gy = 483ADA77 26A3C465 5DA4FBFC 0E1108A8 FD17B448 A6855419 9C47D08F FB10D4B8 n = FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE BAAEDCE6 AF48A03B BFD25E8C D0364141 |

3. **Expanding to 2-2 threshold signature scheme**



It's easy for both parties to calculate public\_key directly and r by a standard [Diffie-Helman key exchange](https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange):

But how to calculate s?

**3.1 Homomorphic Encryption （同态加密）**

Homomorphic encryption is a form of encryption with an additional evaluation capability for computing over encrypted data without access to the secret key. The result of such a computation remains encrypted.

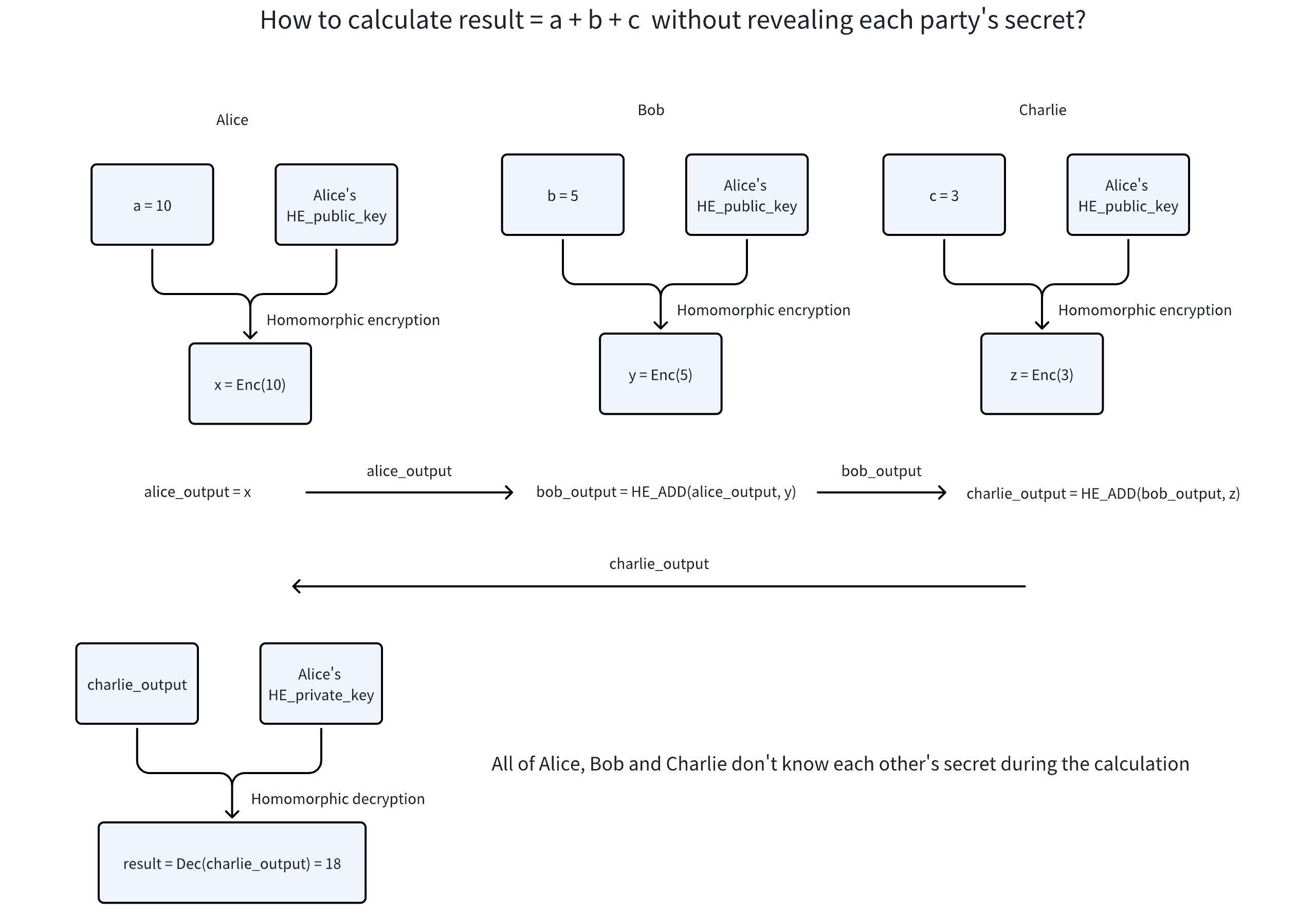
A simple example of homomorphic calculation:

If we define "p" as a public parameter:

, and

Then

And in this case:



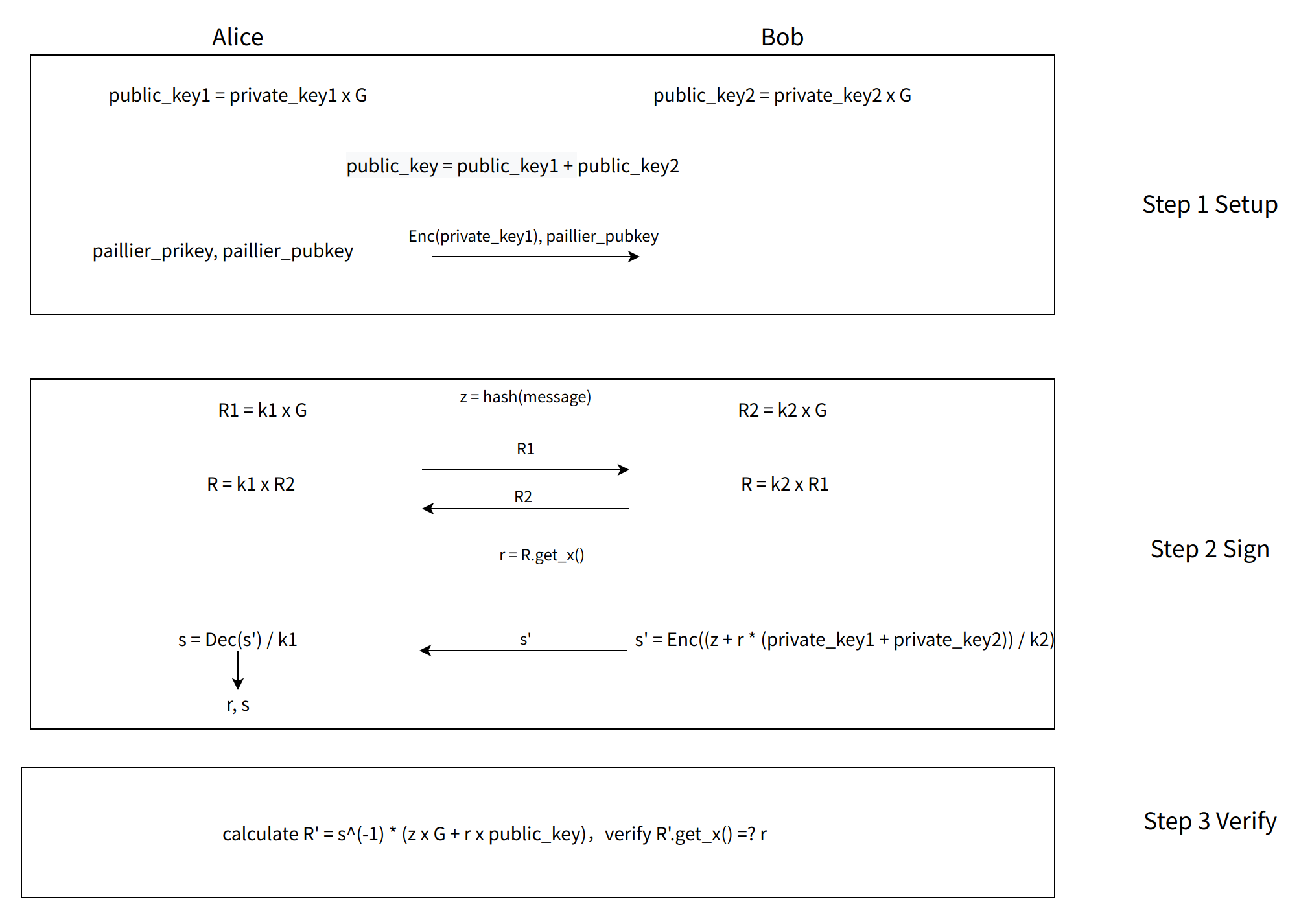
* *Partially homomorphic: encryption* encompasses schemes that support the evaluation of circuits consisting of only one type of gate, e.g., addition or multiplication.
* *Somewhat homomorphic: encryption* schemes can evaluate two types of gates, but only for a subset of circuits.
* *Leveled fully homomorphic: encryption* supports the evaluation of arbitrary circuits composed of multiple types of gates of bounded (pre-determined) depth.
* *Fully homomorphic encryption* (FHE): allows the evaluation of arbitrary circuits composed of multiple types of gates of unbounded depth and is the strongest notion of homomorphic encryption.

[Paillier cryptosystem](https://en.wikipedia.org/wiki/Paillier_cryptosystem) is a typical additive homomorphic encryption system, which is enough for our scenario.

**3.2 Lindell17 protocol**

Lindell17 protocol is an efficient way to calculate a 2-out-of-2 threshold signature.

Target:



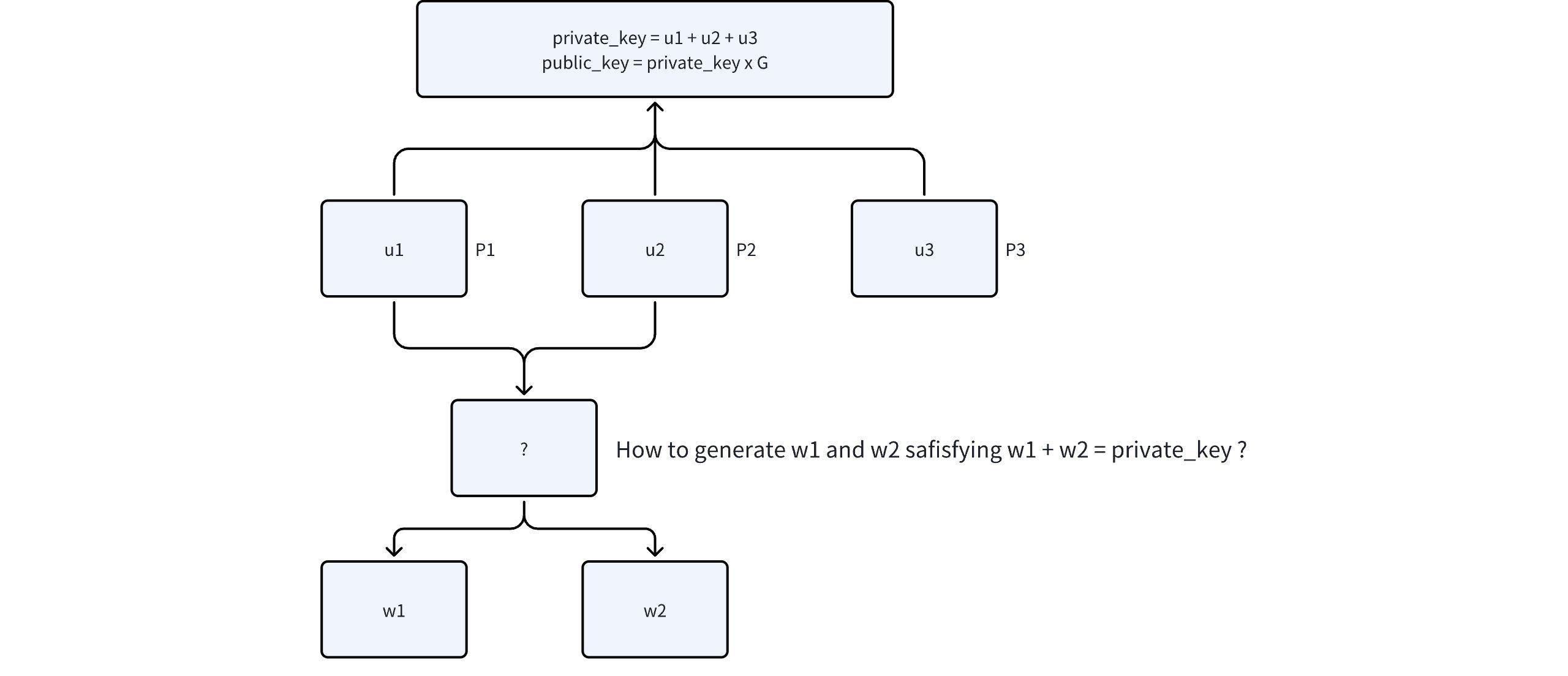
With paillier\_pubkey, Bob could easily calculate s' :

More details can be found at https://www.youtube.com/watch?v=pwc\_Ork-1aA.

4. **Expanding to 2-3 threshold signature scheme**

If we have 3 private key pieces, how can we construct a signature by any 2 of the 3 ?

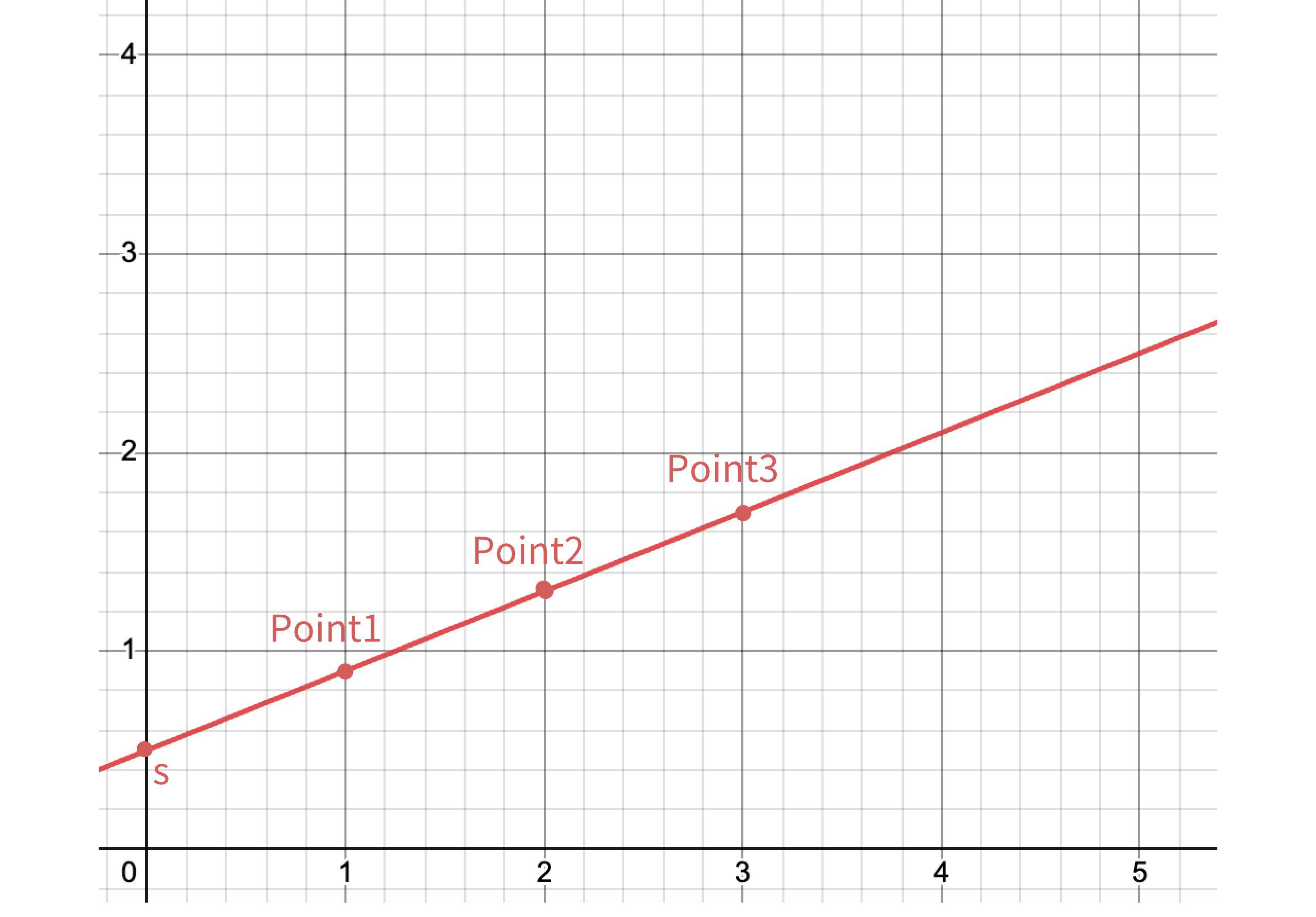
More specifically, how to generate the key pair (private\_key, public\_key)? And if we assign P1 and P2, how to generate secrete sharding w1 and w2, which satisfying w1 + w2 = private\_key ?



**4.1 Shamir Secret Sharing Scheme**

How to split a secret "s" into 3 pieces, so that any 2 of the 3 could recover s ?

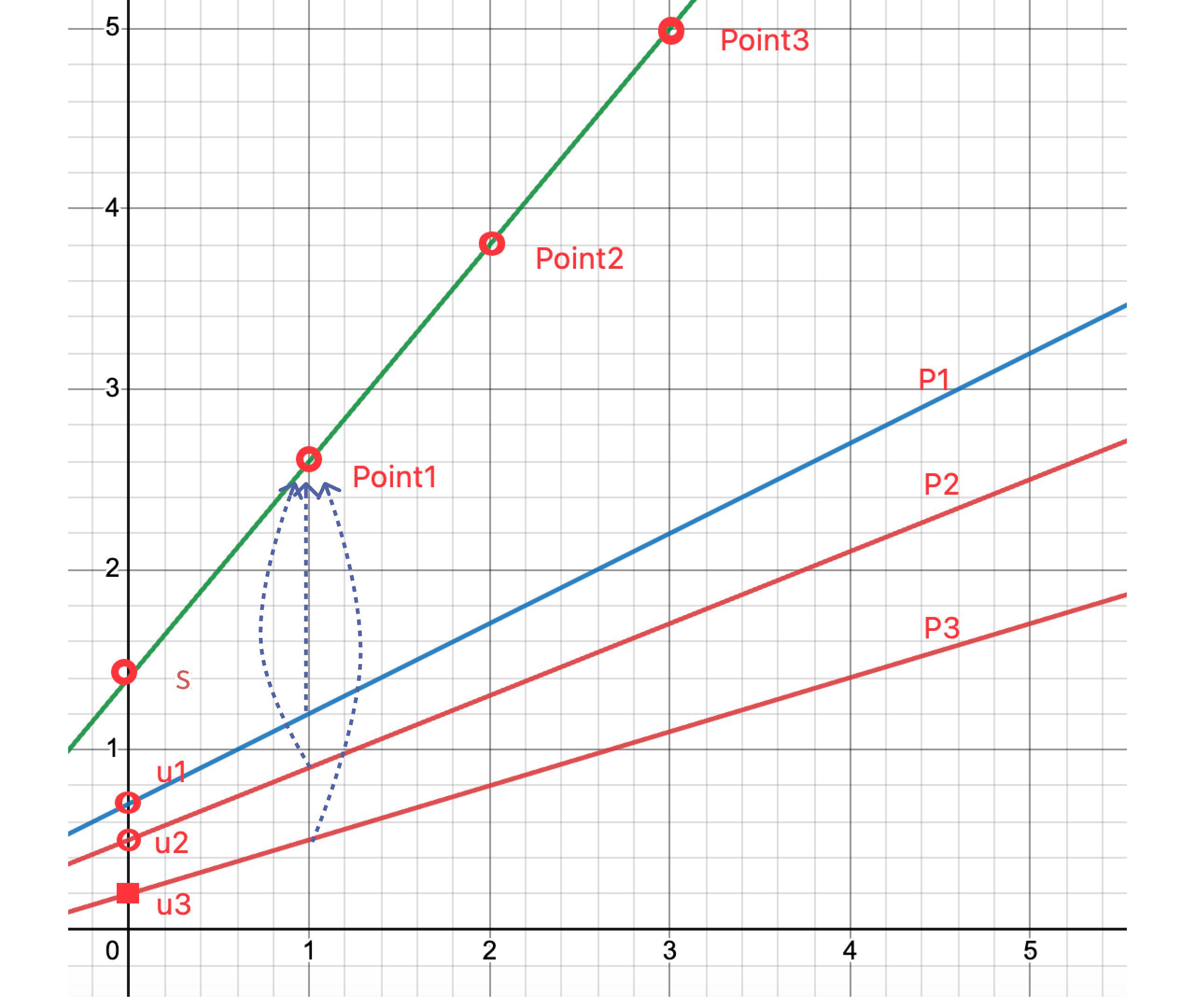
Consider drawing a line across (0, s), then pick 3 points on the line and assign them to the 3 parties, so any 2 of the 3 could recover s.



More details can be found at https://evervault.com/blog/shamir-secret-sharing

**4.2 Distributed Key Generation（DKG）**

What if we don't want "s" ever existed on any device? What if we want to generate a virtual "s" distributedly?



* Step 1:

Eachcreates a secret lineon their own device:

Consider that and s doesn't exist in any device.

* Step 2:

Such that P1 could sum together and get **Point1: (1, f(1))**.

* Step 3:

Similarly, P2 could get **Point2: (2, f(2))** and P3 could get **Point3: (3, f(3)).**

* Step 4:

Suppose that we assign P1 and P2 to sign, we just need to use [Lagrange Interpolation](https://en.wikipedia.org/wiki/Lagrange_polynomial) to recover s by Point1 and Point2.

So that:

f(1) and f(2) are private and l(x) is public.

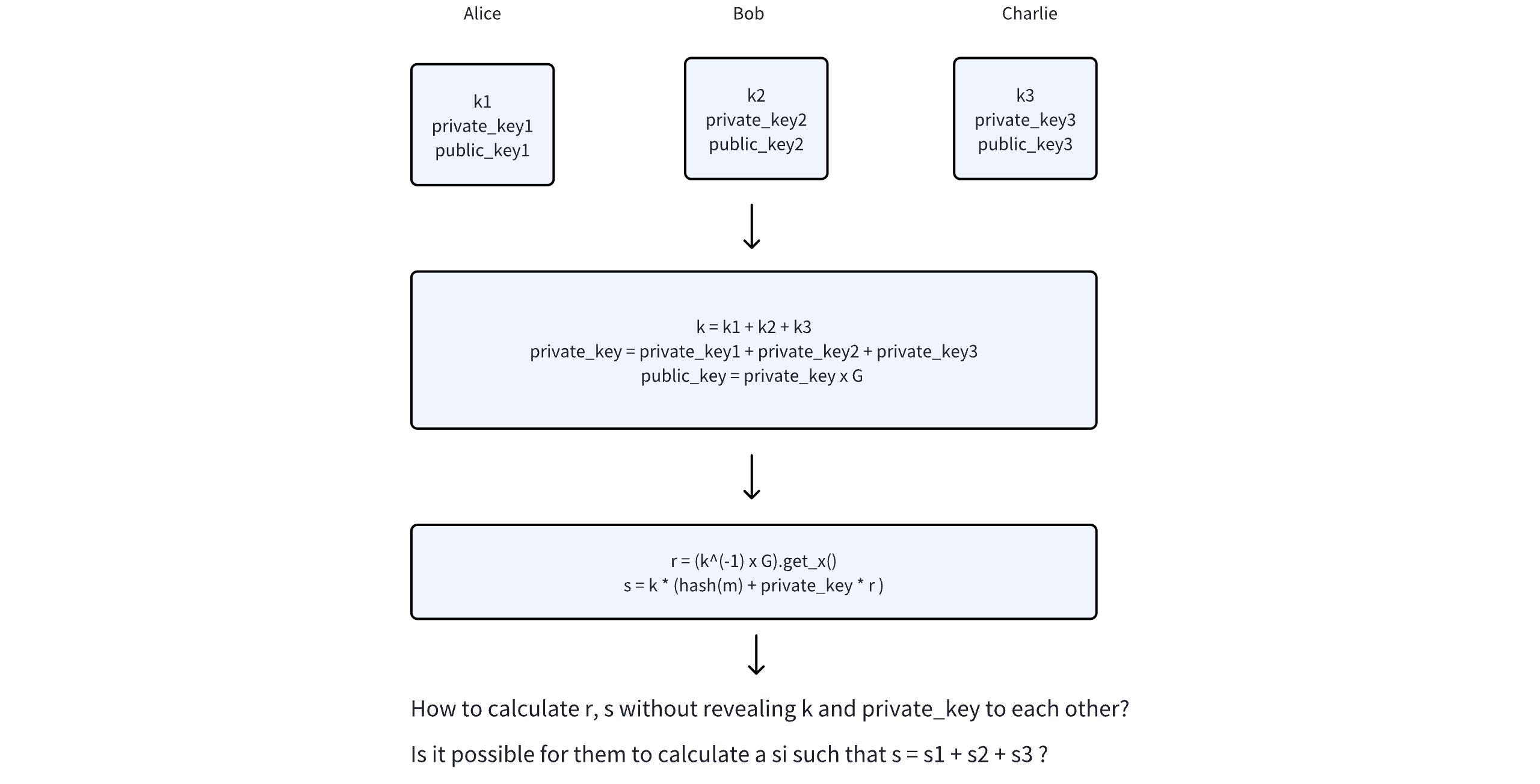
Be noted that is public so

Then we can use Lindell17 protocol to jointly calculate the signature with and

.

5. **Expanding to t-n threshold signature scheme**

What about if we want a further expansion to the t-n threshold signature scheme?



Since there might be numbers of parties, we consider k = k1 + k2 + k3 instead of k = k1 \* k2 \* k3 in order to reduce the bit length of k.

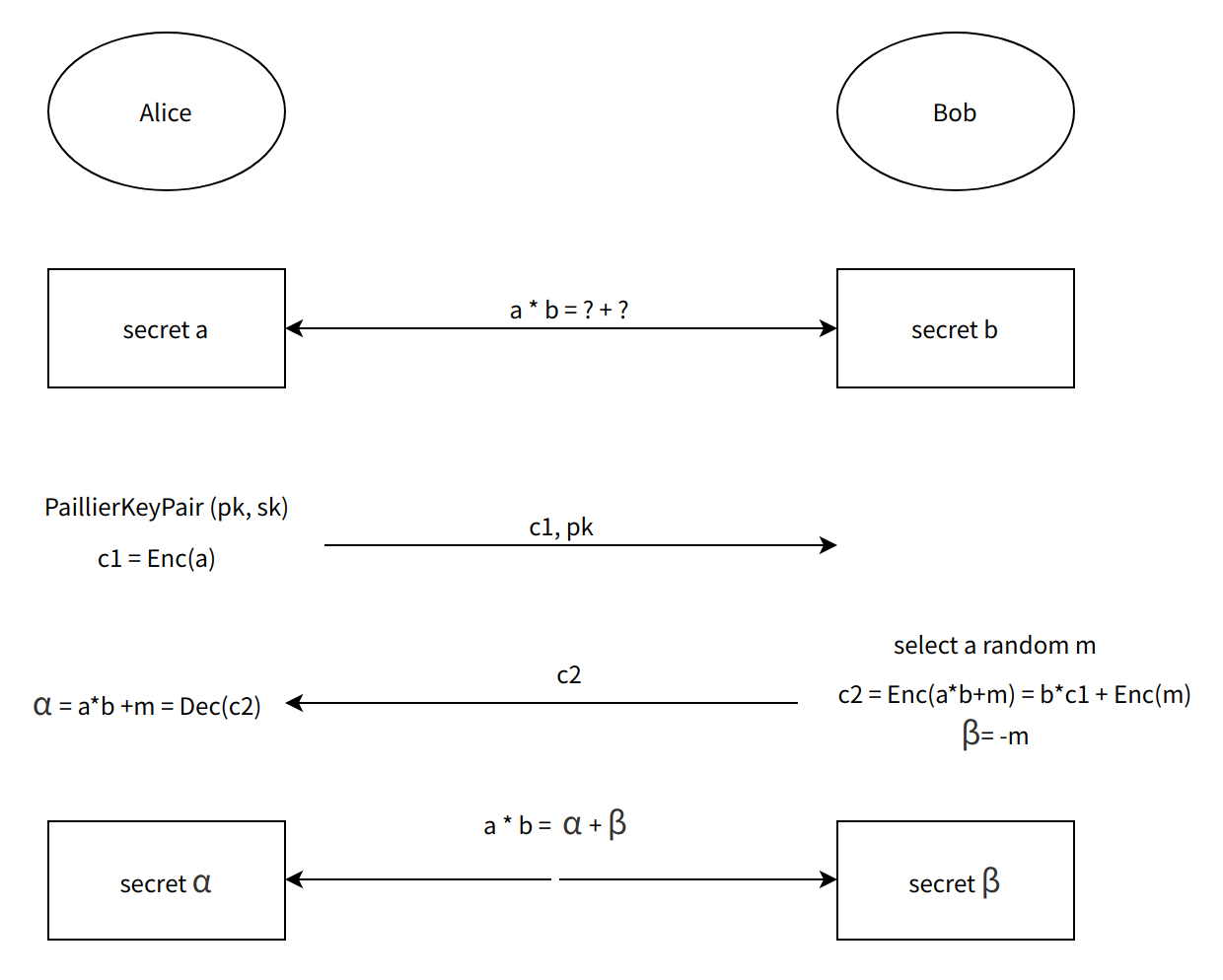
Also, in order to make it easier to cope with, we use "1/k" instead of "k", which is still a random number. Such that

If we could calculate the products like "k1 \* private\_key2", the problem could be solved.

**So how to calculate the product of two secrets without revealing the secrets to each other?**

**5.1 Multiple to Add (MtA)**

By using homomorphic encryption, we can change a MUL operation to an ADD operation.



**5.2 GG18/GG20 protocol**

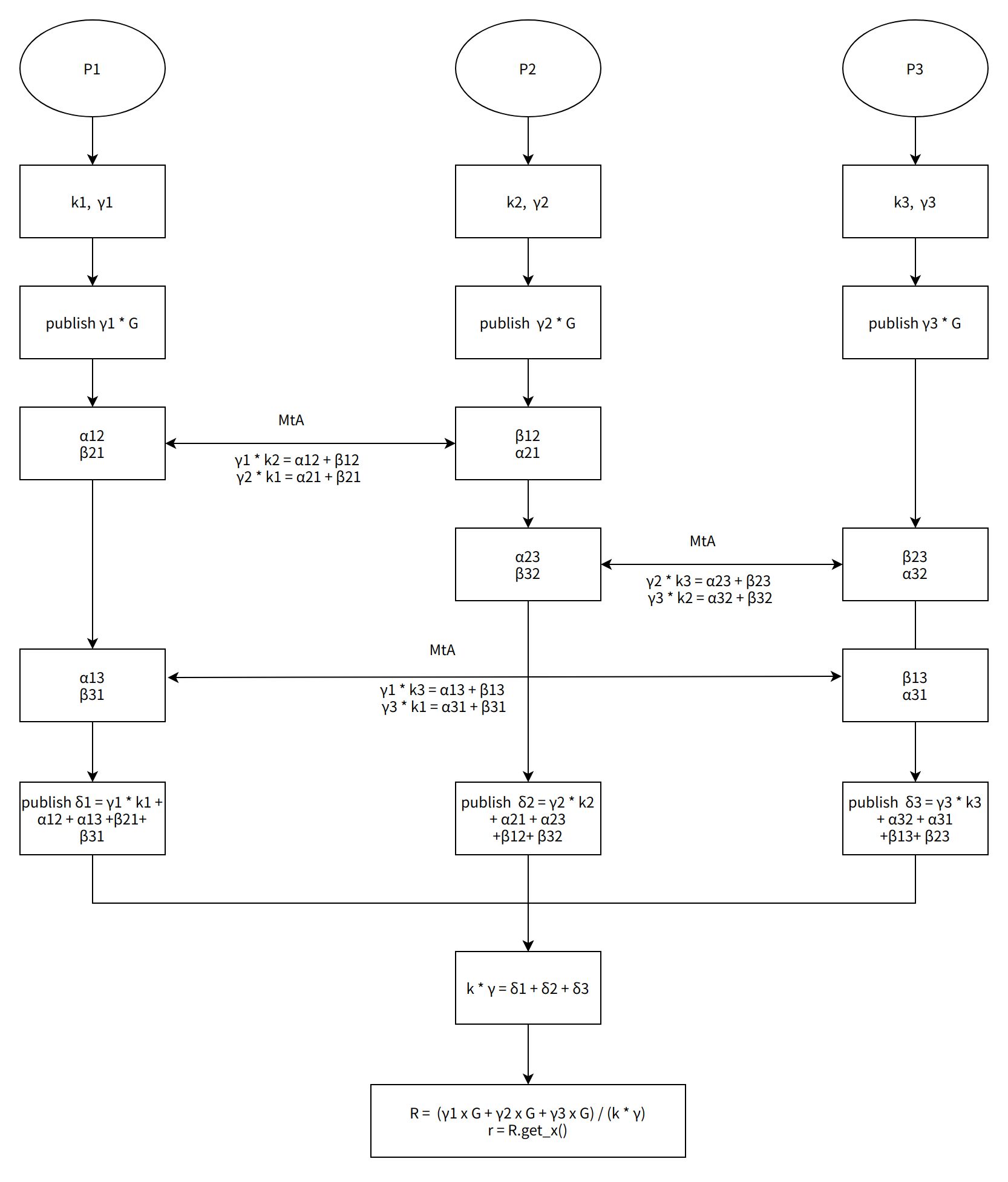
**5.2.1 Calculate r**

* Target:
* Solution

Each party selects another random number γi，so that:

For , each party could calculate and publish it, so that:

For, we use MtA to calculate each product of them.



**5.2.2 Calculate s**

* Target:
* Solution:

can calculateby MtA similarly with the calculation of r.

Similarly, we can calculate and , such that:

6. **Powerful ZKP supporting security**

Recap that the EC operation is in a finite field with modulus p, and be noted that the Paillier Homomorphic operation is in another finite field with modulus N.

For example, if we want the following equation to hold:

We need to guarantee that a \* b will never wrap around N, otherwise the equation will be broken, which might be used for malicious attacks.

So we need to use a large N, for instance N > q^3, and we need ZK range proofs to ensure a\*b < some t (<N) for security.

More information could be found in reference papers and[ZK Solutions to BitForge attack of MPC wallet](https://okg-block.larksuite.com/docx/Bc2QdTf1EolGXWx0BdLuWeWYsJR).

7. **Summery**

* Introduction of threshold signature and OKX MPC wallet scheme
* ECDSA
* 2-out-of-2 threshold signature
* 2-out-of-3 threshold signature
* t-out-of-n threshold signature
* A brief mention of ZKP in MPC wallet

8. **References**

[MPC 技术分享](https://okg-block.larksuite.com/docx/J9nUddJQvoqSVWxFRvduajO9sad)

[ECDSA & MPC](https://okg-block.larksuite.com/docx/doxusPrm4pqEQy3eb6VGAnCIvwb)

[web3 wallet MPC钱包技术实现讨论](https://okg-block.larksuite.com/docs/docusLzAESkj3jV1XaTDTZZ8ypR)

[ECDSA 门限签名算法](https://okg-block.larksuite.com/docx/doxusEvA5gyWWTFqQr2cutdotvd)

https://eprint.iacr.org/2020/540.pdf

https://eprint.iacr.org/2017/552.pdf

https://dl.acm.org/doi/10.1145/3243734.3243859

https://www.youtube.com/watch?v=pwc\_Ork-1aA

[ZK Solutions to BitForge attack of MPC wallet](https://okg-block.larksuite.com/docx/Bc2QdTf1EolGXWx0BdLuWeWYsJR)