Ring Signature

In a ring signature scheme, the signatures produced by different group members look indistinguishable to their verifiers. Ring signatures are useful when the members do not want to cooperate. In a ring signature scheme, there are no prearranged groups of users, there are no procedures for setting, changing, or deleting groups, there is no way to distribute specialized keys, and there is no way to revoke the anonymity of the actual signer (unless he decides to expose himself). Our only assumption is that each member is already associated with the public key of some standard signature scheme such as RSA. To produce a ring signature, the actual signer declares an arbitrary set of possible signers that includes himself, and computes the signature entirely by himself using only his secret key and the others' public keys.

Working

The assumption is that each possible signer is associated with a public key $\boldsymbol{P_k}$ and the corresponding secret key is denoted by $\boldsymbol{S_k}$. A ring signature scheme is set-up free: The signer doesn't need knowledge, consent or assistance of the other ring members to put them in the ring – the only thing is required are their regular public keys. The two procedures involved in this scheme are as follows:

- **Signing:** This produces a ring signature σ for the message m, given the public keys $\boldsymbol{P_1},\ \boldsymbol{P_2},\ \boldsymbol{P_3}, \dots ,\ \boldsymbol{P_{size}}$ where size is the number of members in the ring/group, together with the secret key $\boldsymbol{S_s}$ of the actual signer.
- **Verify:** This accepts the message m and a ring signature σ, outputs either true or false.

Explaining the implementation

Consider a group with four members and each have their own public key and secret key. If a participant wants to sign a message from the group, then initially, they have to generate a random value **u** and compute **v**, and then generate random values $\boldsymbol{S_i}$ for each of the other participants and take their own secret key which will reverse the encryption function. Each of the random values of the other participants are encrypted and the signer computes their own value such the result of the ring must be equal to **v**.

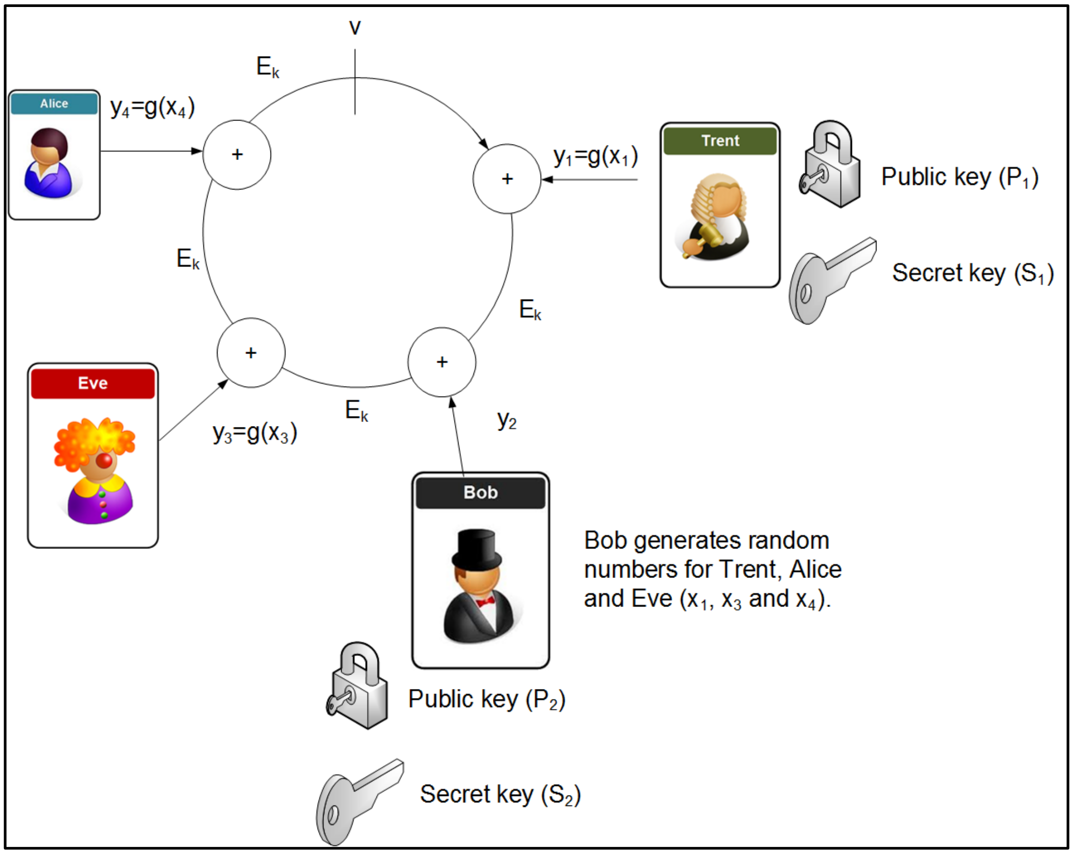

Image credit: https://asecuritysite.com/encryption/ring_sig

The **sign()** and **verify()** function in the python file are explained below:

sign()

The hash of the message is computed, let's say this result to be **p**. A random initial value **u** is generated and is encrypted using the message hash, lets say this to be **v** = $\boldsymbol{E_p(u)}$. Now, for each person (apart from the actual signer), we calculate **e** which is encrypted using naïve RSA ($x^e \ mod \ n$ where x = random number, e is the public key of a participant). We then XOR this value such that **v = v ^ e**. After this is done for each participant in the group, we move to the actual signer. The actual signer then signs it using their secret key and then returns the signature list containing the original value of **v** and the values generated.

verify()

The verify function starts by computing the hash of the message, denoted as **p**. The first element of the signature list is then assigned to **c**, which is the initial value of the signature. Next, for each participant in the ring, the function computes an intermediate value **e** using the public key and the value X[i+1] from the signature, where **e** is calculated as (X[i+1]^e) mod n. This is done for all participants in the ring, generating a list of y values. The function then iteratively computes a chained hash by using naïve RSA and XORing each intermediate value y[i] with the previous result, starting with **c**. After the loop completes, the final value of the chain is compared with the initial value c. If they match, it confirms that the signature is valid and the message has not been tampered with, returning True. If they do not match, it indicates that the signature is invalid, and the function returns False.

Sources

https://asecuritysite.com/encryption/ring_sig

https://link.springer.com/chapter/10.1007/3-540-45682-1_32