PRIVACY: 2.0

CHAP 2:

Idea: Operations on encrypted data making the decrypted result as the wanted aggregate.

→ Each voter can check his vote is unmodified. VERIFIABILITY

→ Each voter can compute the results (tally). AGGREGATE VISIBILITY

→ Each voter can is the only one who knows for who he voted. PRIVACY

We need a ~~trade~~ Trade-off VERIFIABILITY / PRIVACY

Trust independence towards voters and institutions.

Distributed Homomorphic Public Key Encryption:

(ELGAMAL/RSA)

→ Public Key Encryption: Anyone can encrypt a message with the public key but only the holder of the private key can decrypt it.

→ Homomorphic Encryption: Allows computations on encrypted data without decrypting it. $Enc(n_1) \times Enc(n_2) = Enc(n_1 + n_2)$

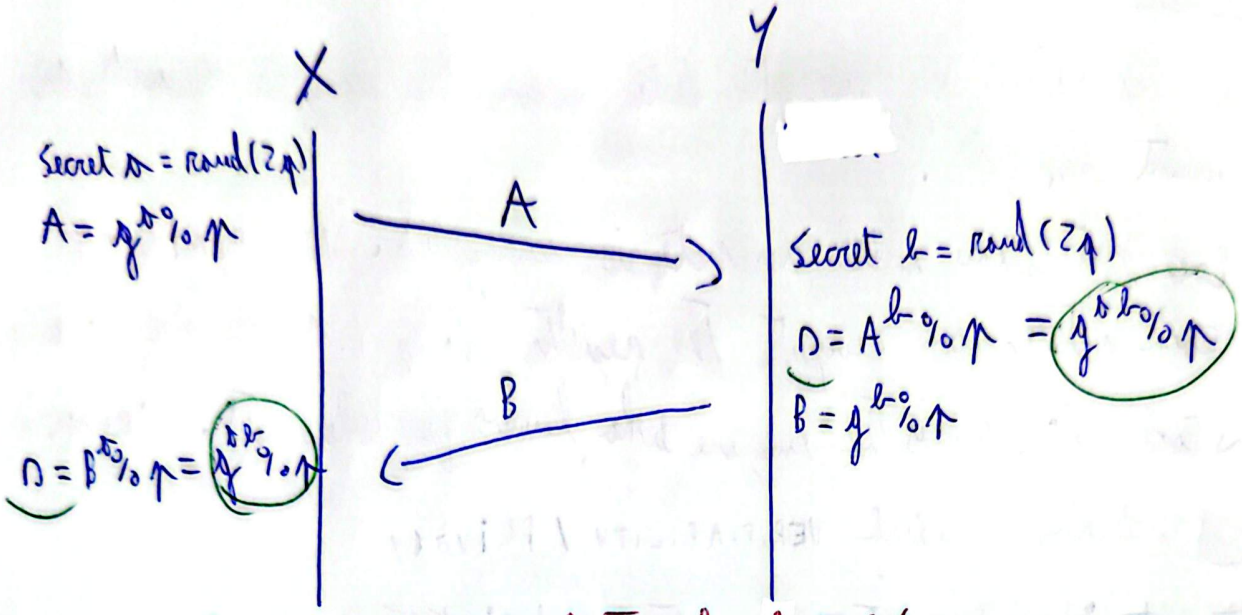
→ Distributed key management: Private key split across different parties. Parties must collaborate to be able to decrypt.

So: 1) There is a public key everyone can use to encrypt data.

2) The encryption is homomorphic, so computations can be done directly on ciphertexts.

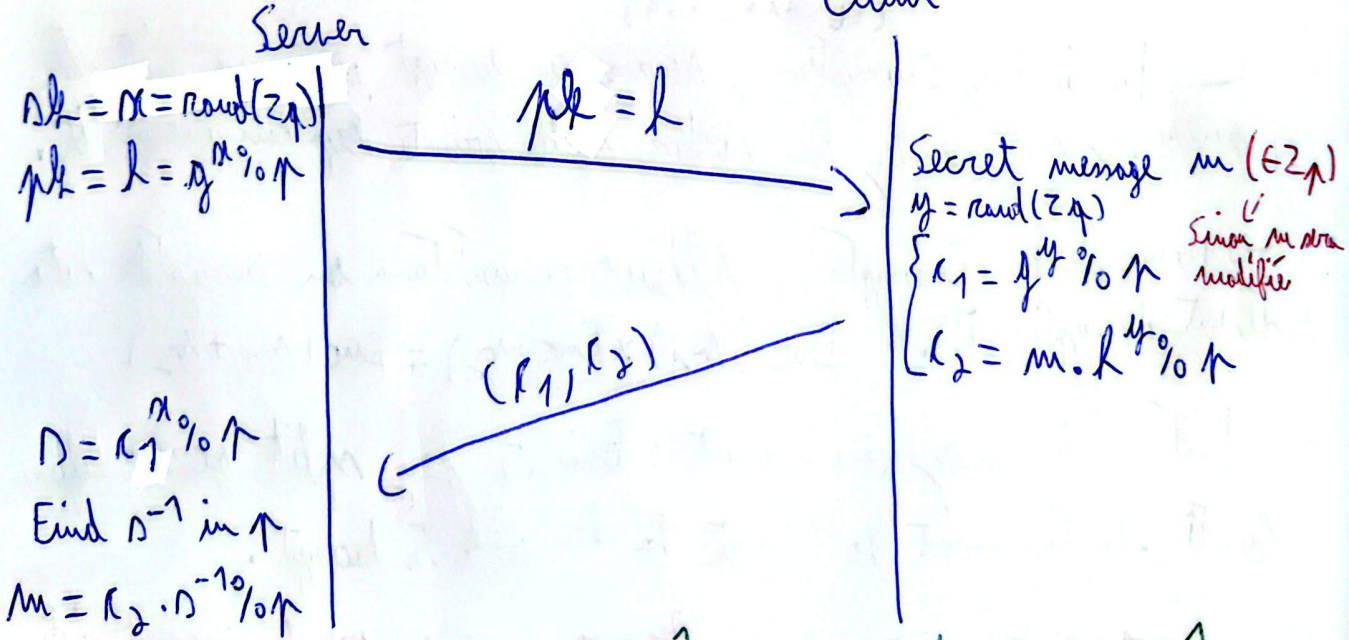
3) The decryption is distributed ~~among~~ among multiple parties, so only a collaboration of them can decrypt the result.

Diffie - Hellman : Agreement on a shared secret key.



*Only a discrete log can break the algo but it's NP-HARD to compute.

EL GAMAL : Public Key Encryption



$$L_3 = m \cdot h^y \cdot [(g^y)^x]^{-1} = m \cdot \underbrace{(g^x)^y \cdot [g^{xy}]^{-1}}_{=1} = m$$

Again *!!

Eternat's Little: Efficiently find the inverse of a number in a modulo group.

$((\cdot) \cdot z_p) \,\%\, p = z_p$

$(\cdot)$ → constant; $p-1$ elements

$(\cdot z_p = \{(\cdot 1, (\cdot 2, \dots, (\cdot (p-1)\}$ $(\%\, p)$

Remember: $a, b \in z_p \Rightarrow (a \cdot b) \,\%\, p = r \in z$

So: $\left(\prod_i z_p[i]\right)^{\%\, p} = \left(\prod_i (\cdot z_p[i]\right)^{\%\, p} \equiv$ As (has an inverse in z_p (p is prime) this multiplication is modulo p is just a permutation.

$\hookrightarrow = ((1 \cdot ()) \times (2 \cdot ()) \times \dots \times ([p-1] \cdot ())) \,\%\, p$

$= (p-1)! \cdot (1, 2, (p-1)) \,\%\, p$

$= (p-1)! \cdot (p-1)! \,\%\, p$

$= (p-1)! \cdot ((p-1)! \,\%\, p) \,\%\, p$

$\Vert \quad k \in z_p$

$\Leftrightarrow (p-1)! \,\%\, p = (p-1)! \cdot (p-1)! \,\%\, p$

$\Leftrightarrow 1 \,\%\, p = (p-1 \,\%\, p$

$\Leftrightarrow 1 \cdot (^{-1} \,\%\, p = (p-1 \cdot (^{-1} \,\%\, p$

$\Leftrightarrow$ $\boxed{(^{-1} \,\%\, p = (p-2 \,\%\, p}$

Expo El Gamal: for messages $m \in \{0;1\}$

Simple ~~El Gamal sec~~

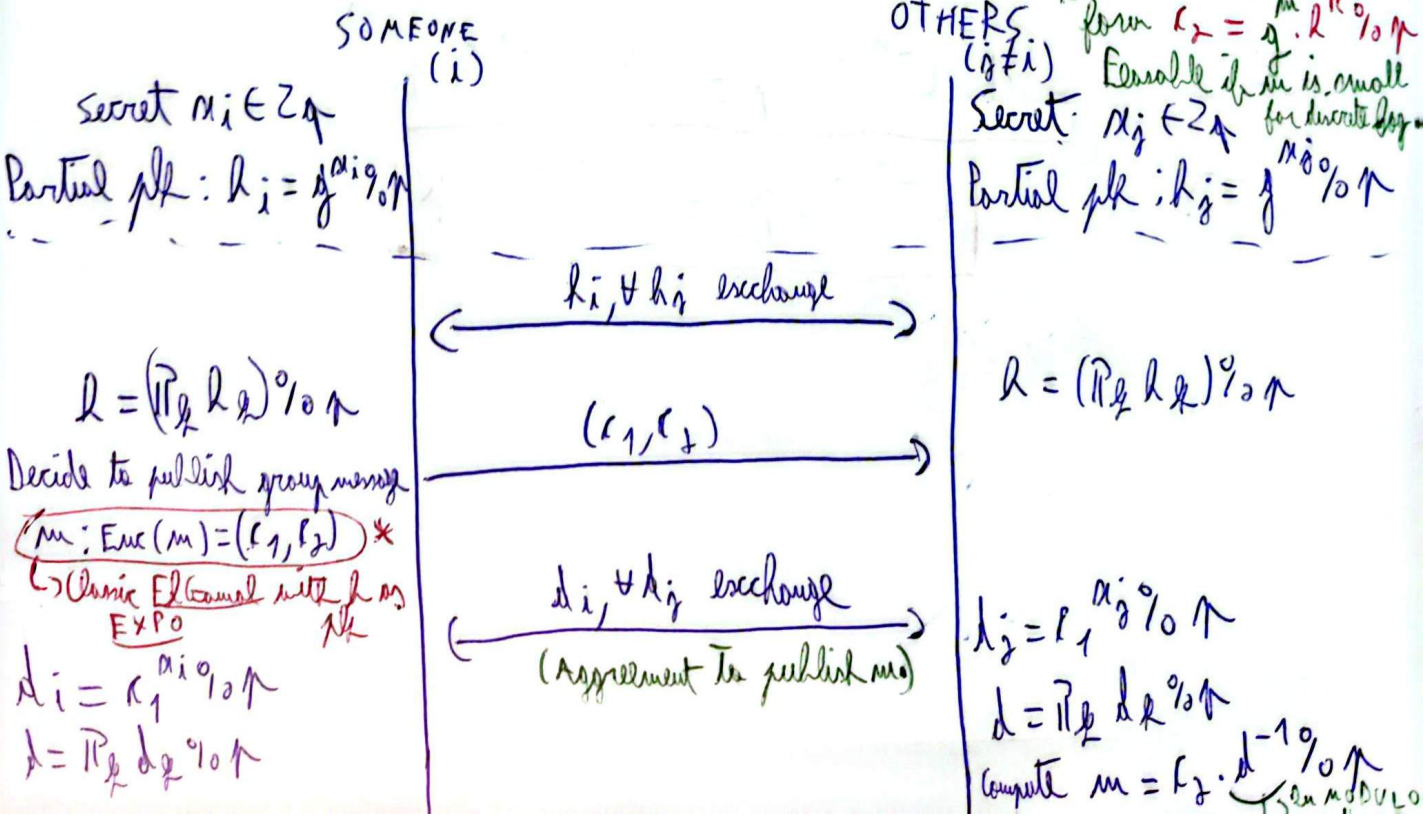
Classic ElGamal + $m' = g^m \% p \mid m \in \{0,1\}$

Distributed Key Generation ElGamal:

In DKG for ElGamal, a group of ~~parties jointly~~ peoples computes together one shared public key, and the private key is split into shares.

Only a threshold of participants working together can decrypt a ciphertext. An ~~all~~ individual participant cannot decrypt alone with their own share.

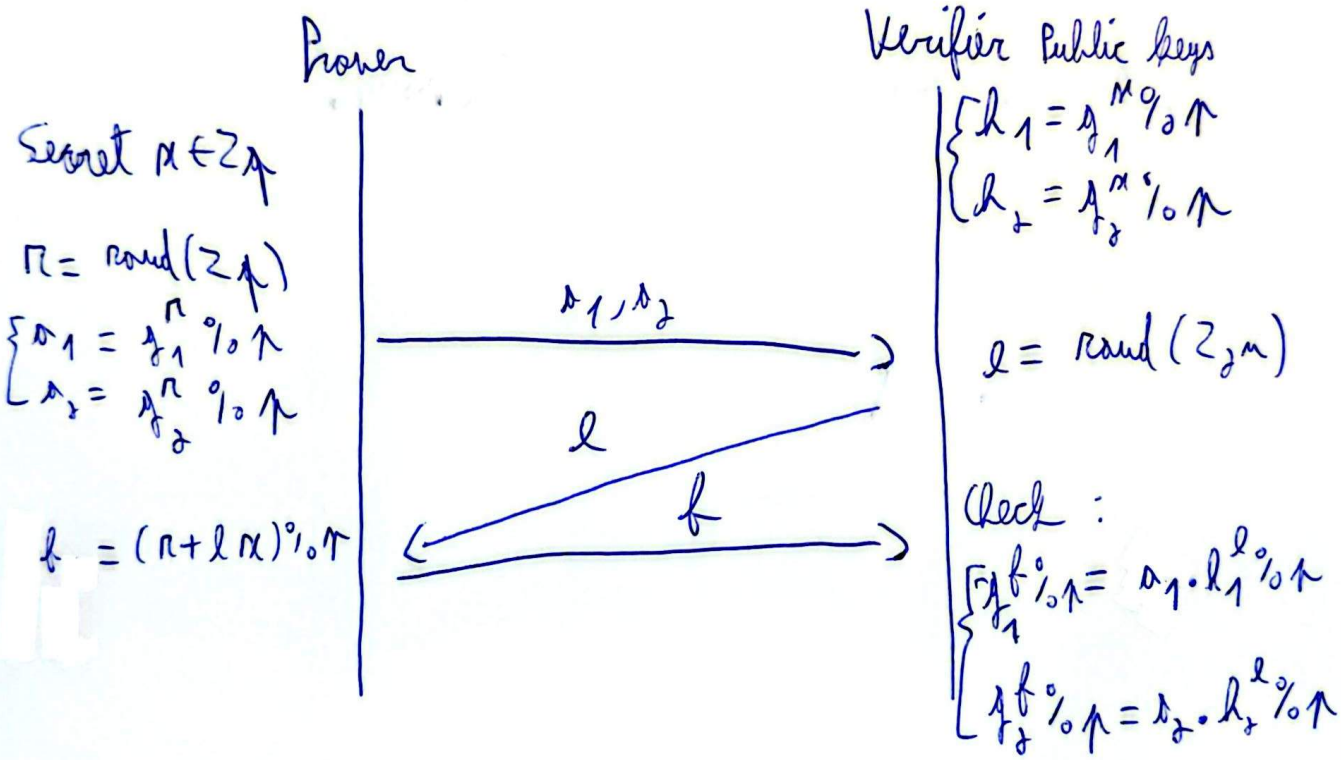
Once all trustees publish their partial decryptions, any outsider can recover the message. Only security here is joint consent but there is not secrecy after consent.



Chaum-Pedersen:

Schnorr proves knowledge of a discrete log.
Chaum-Pedersen proves equality of discrete logarithms. → Chaum-Pedersen is a stronger proof.
Both without revealing x.

Completeness: If a user is legitimate, he'll always succeed maths.
Soundness: ~~If I can prove~~ 5.32

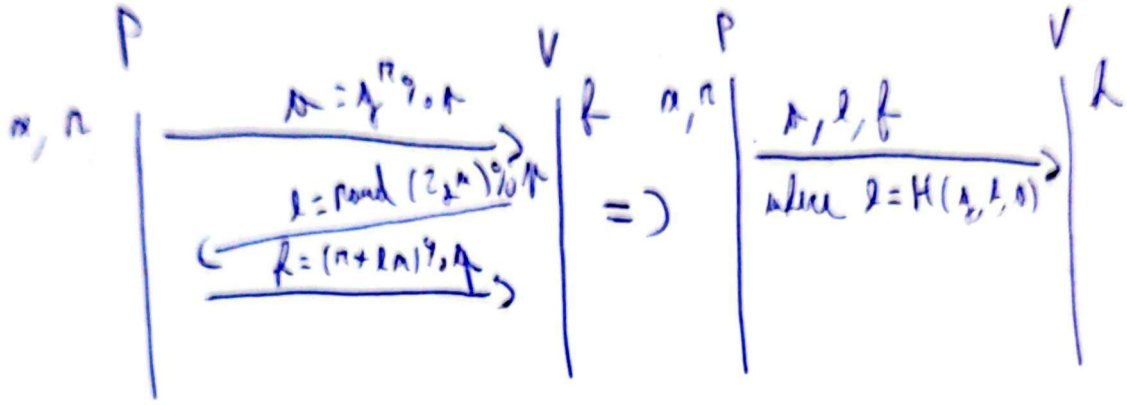


Two groups G_1 and G_2 have different g_i but common p and q.

From $G = (g, p, q)$ you can find $G' = (g', p, q)$ as:
$r = rand(\mathbb{Z}_q^*) \quad (0 < r < q) \wedge gcd(r, q) = 1$
$g' = g^r \% p$

Non-Interactive Zero-Knowledge Proof:

Idea: The prover computes a predictable challenge.



Challenge e becomes an hash function value.