X_1

transm

, X_2, ... X_N and Y_1, ..., Y_N: these are not

itted.

$r = r_1 + r_2$ a

forc_k for $k =$

Each m-dimen

generated by h

$c_k \in \{-1, 0$

as random seed

$1 \ldots N$

nsional c_k is

hashing r, giving

$0, 1\}^m$

for each

5.1 $User$

5.2 Use

5.3

$User$

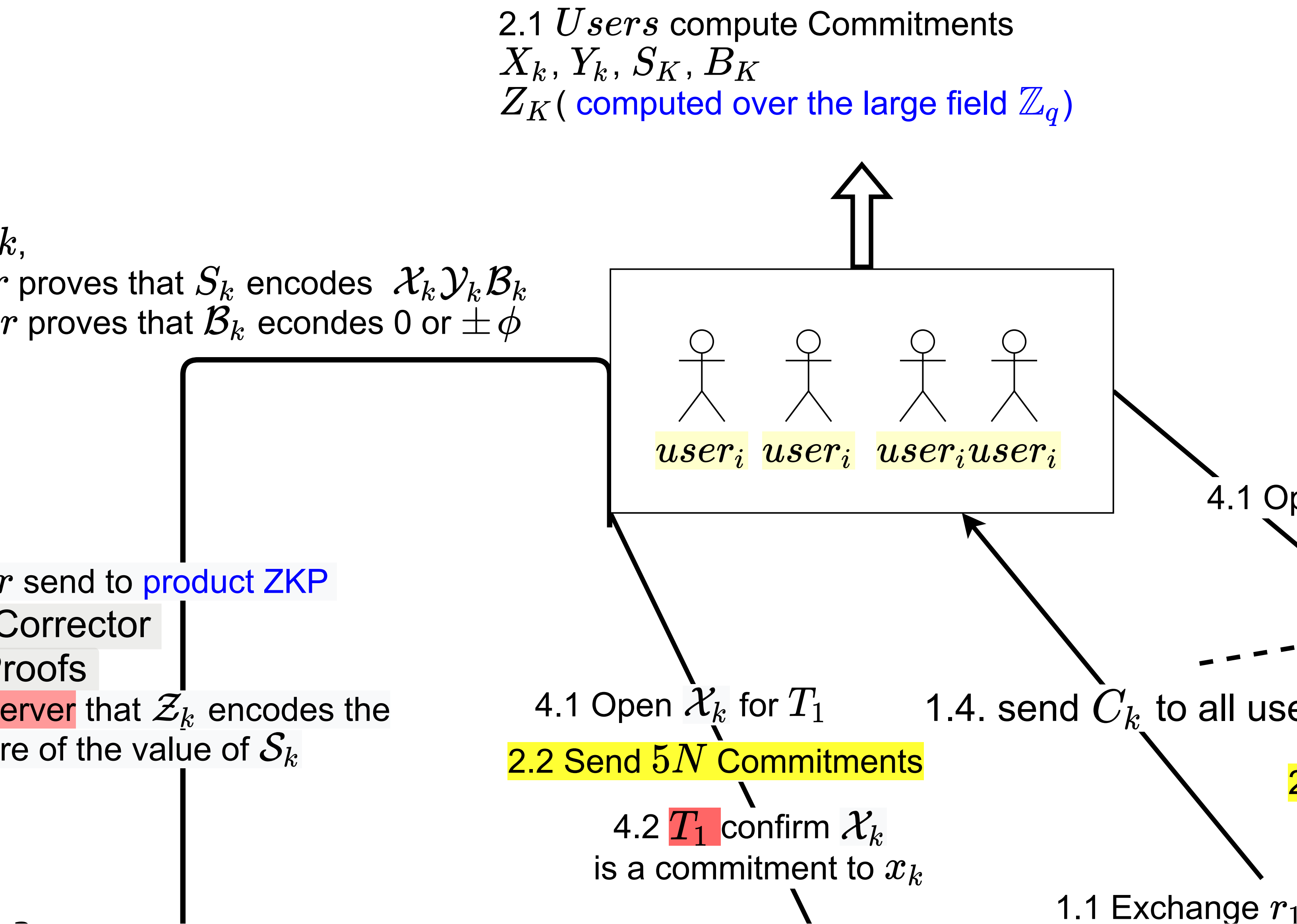
mdC

scP

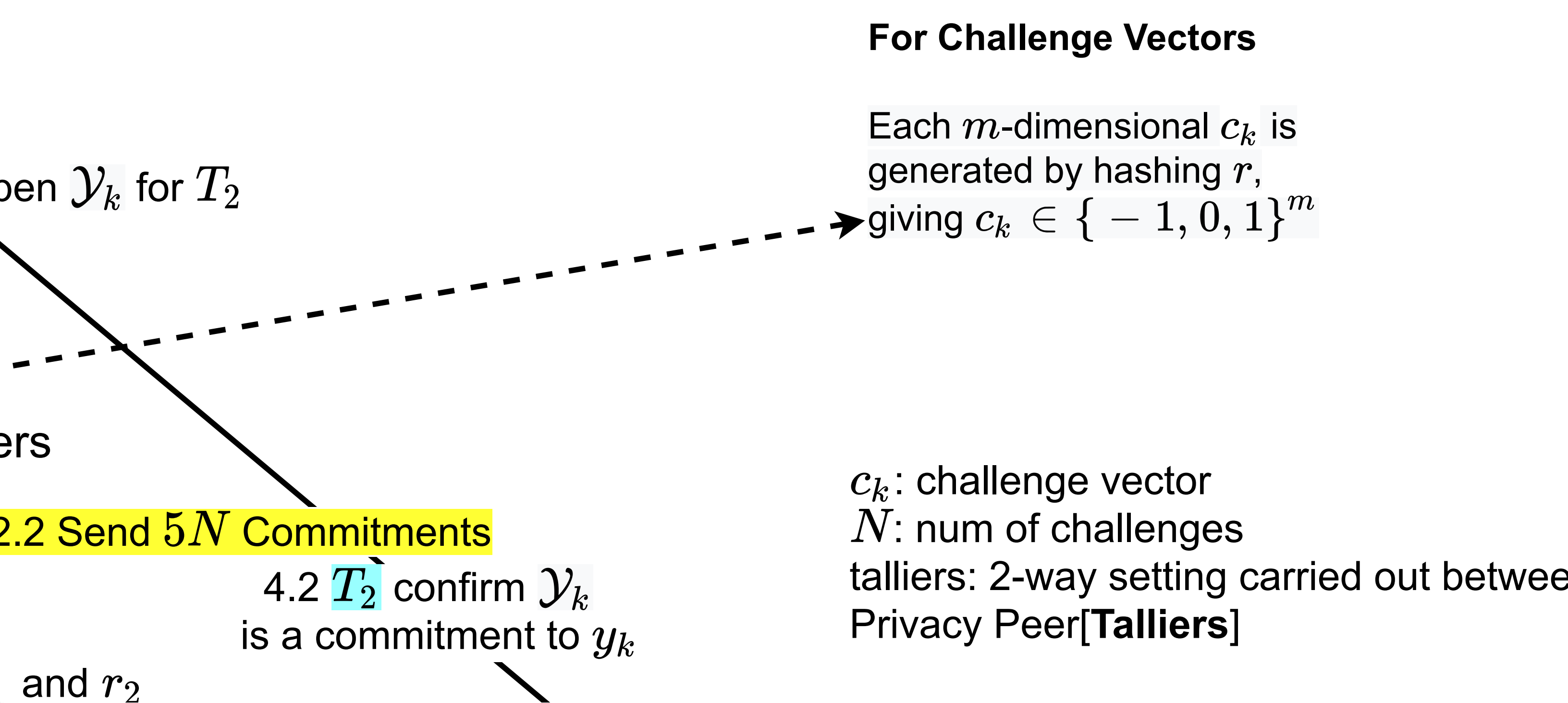
to s

squa

scProofs contains commitment to



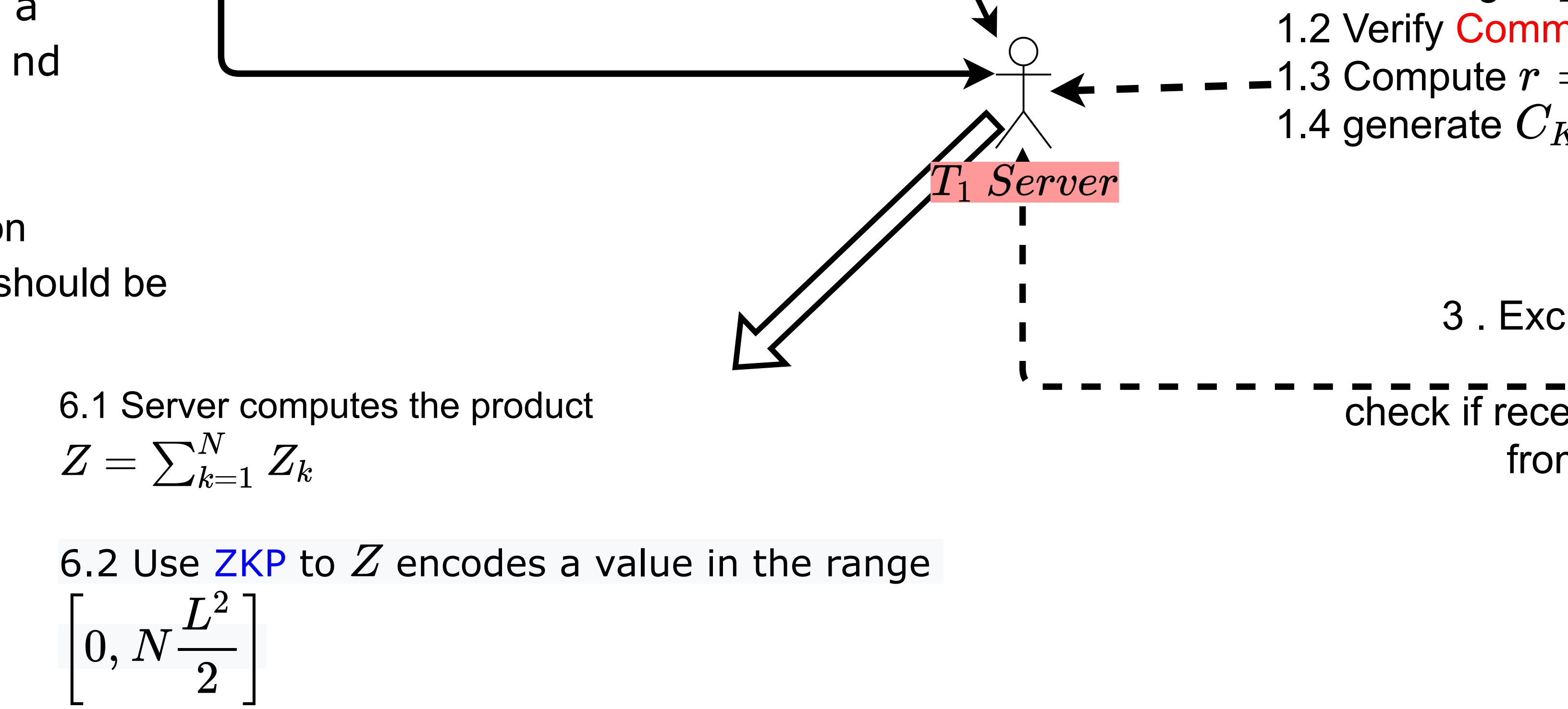
2.1 $Users$ compute Commitments
X_k, Y_k, S_K, B_K
Z_K (computed over the large field $\mathbb{Z}_q$)
k,
r proves that S_k encodes $\mathcal{X}_k \mathcal{Y}_k \mathcal{B}_k$
r proves that $\mathcal{B}_k$ econdes 0 or $\pm\phi$
$user_i$ $user_i$ $user_i user_i$
4.1 Op
r send to product ZKP
Corrector
roofs
erver that $\mathcal{Z}_k$ encodes the
re of the value of $\mathcal{S}_k$
4.1 Open $\mathcal{X}_k$ for T_1
2.2 Send $5N$ Commitments
4.2 T_1 confirm $\mathcal{X}_k$
is a commitment to x_k
1.4. send C_k to all use
1.1 Exchange r_1

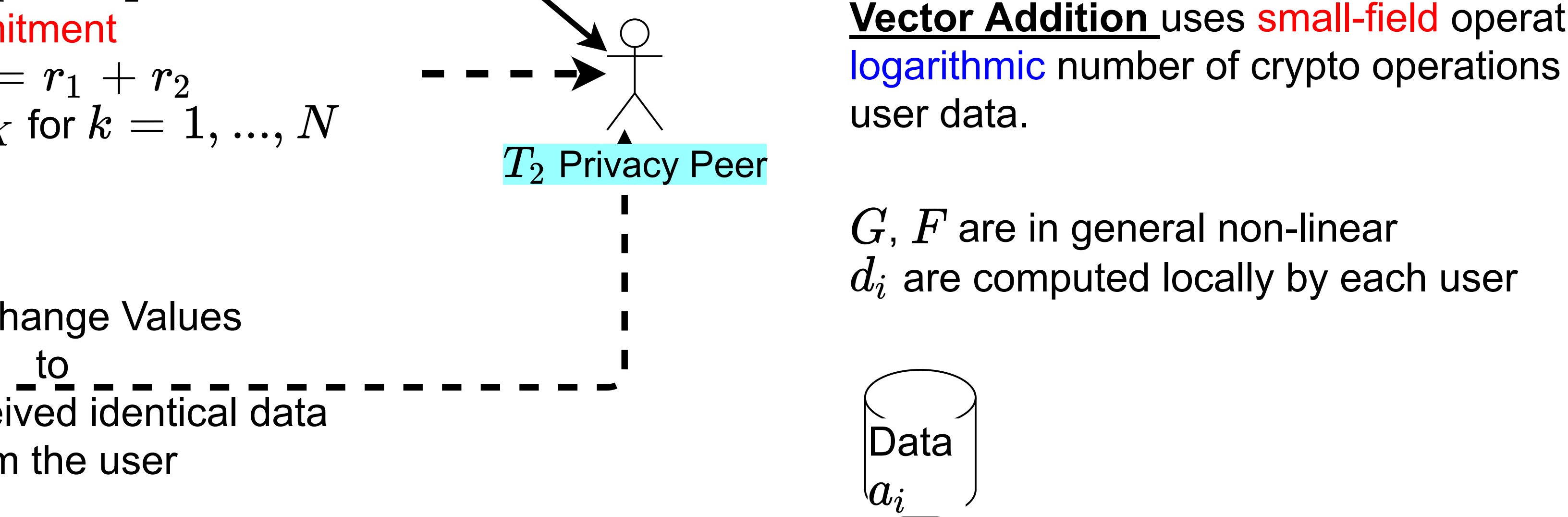

For Challenge Vectors
Each m-dimensional c_k is generated by hashing r, giving $c_k \in \{-1, 0, 1\}^m$
pen $\mathcal{Y}_k$ for T_2
ers
2.2 Send $5N$ Commitments
4.2 T_2 confirm $\mathcal{Y}_k$ is a commitment to y_k
and r_2
c_k: challenge vector
N: num of challenges
talliers: 2-way setting carried out betwee
Privacy Peer[**Talliers**]

en server and

scProofs contains commitment to
number and square so both S_k *
`Z_k` are stored

mdCorrector: The modular reductio
corrector (the B's in the paper). They s
the commitment to 0 or +/-F.


a
nd
on
should be
1.2 Verify Comm
1.3 Compute r =
1.4 generate C_K
T_1 $Server$
3 . Exc
check if rece
fror
6.1 Server computes the product
$Z = \sum_{k=1}^{N} Z_k$
6.2 Use ZKP to Z encodes a value in the range
$\left[0, N\frac{L^2}{2}\right]$



itment
$= r_1 + r_2$
for $k = 1, \ldots, N$
T_2 Privacy Peer
hange Values
to
eived identical data
m the user
Vector Addition uses small-field operat
logarithmic number of crypto operations
user data.
G, F are in general non-linear
d_i are computed locally by each user
Data
a_i

ions
in the size of

Commitment

~ g: NativeBigInteger
~ h: NativeBigInteger
~ val BigInteger
~ r BigInteger // randomness used in the commitment

+ sanityCheck(): void
+ Commitment(g, h)

~ computeCommitment(val, r) BigInteger
+ commit(long val) BigInteger
+ commit(BigInteger val)BigInteger

// commit to Z_q using given randomness
+ commit(BigInteger val, BigInteger r)
+ getRandomness() BigInteger
+ getValue() BigInteger

+ verify(BigInteger c, BigInteger val, BigInteger r)// check