EX3 RDS 2026
Payment Scheme
Version 1

System model. Assume a system of n servers communicating synchronously, with known bounded message delay, Δ. Assume $n = 2f + 1$, and that at most f servers may suffer omission failures (they may fail to send messages, and or fail to receive messages, but never send incorrect messages).

Assume a set U of users, some of which may be malicious. Each user $u \in U$ has a public key pk_u and a corresponding secret key sk_u.

1 Payment Scheme

Signature model. Assume a signature scheme of the following abstract form. The key generation algorithm creates a pair (sk, pk). The signing algorithm is *deterministic*: given a message m and a secret key sk, it always produces the same signature.

To sign a message m under a secret key sk, the signing algorithm first hashes the message into an appropriate algebraic group using a public hash function H, and then applies a group operation parameterized by the secret key. Concretely, a signature has the form

$$\sigma = \mathsf{sign}(m, sk) = H(m)^{sk}.$$

Verification checks that the signature corresponds to the public key pk associated with sk. This abstraction captures common signature schemes where the secret key acts as an exponent and avoids any reliance on signing randomness.

In our setting, the system wide public key is PK which is know to all parties and the signing key is $sk = g(0)$, where g is the shared polynomial.

A token (P, v, σ) is therefore valid exactly when the servers collectively authorize it by producing or validating a signature under the shared secret $g(0)$.

Assume each server i holds a share $s_i = g(i)$, where g is a uniformly random polynomial of degree at most f, and $g(0)$ is a secret shared among the servers.

A *token* is a triple $T = (P, v, \sigma)$, where P is a public key, v is a non negative integer value, and σ is a signature.

A token $T = (P, v, \sigma)$ is *valid* if its signature is a valid signature on its public key and value relative to the system wide public key PK:

$$\mathsf{verify}((P, v), \sigma, PK) = 1.$$

A token is considered *unspent* if it has not been previously consumed by a successful payment operation.

In addition, the system stores for each user the value of its un minted balance. Assume that initially each user has an un minted balance of 100.

Using this setup, design a replicated payment scheme supporting the following three operations:

Mint

A user u creates a new key pair (sk, pk) and submits a request containing (pk, v, σ) where $\sigma = \mathsf{sign}((pk, v), sk_u)$. Upon receiving such a request, the servers check that v does not exceed the current un minted balance of user u. If so, they decrease the un minted balance by v and collectively issue a fresh valid token

$$T = (pk, v, \sigma),$$

provided the request is authorized relative to the public key pk_u.

Interactive Pay

This operation allows a user to pay another user and optionally combine two tokens and receive change. It requires the recipient to provide a public key P_a.

A user submits two valid unspent tokens T_1, T_2, together with a transaction request specifying output public keys and values (P_a, v_a) and (P_b, v_b). The servers verify that $v_1 + v_2 = v_a + v_b$ and that the whole request is properly

authorized by signatures under the public keys associated with T_1 and T_2. If so, the servers atomically mark T_1 and T_2 as spent and issue two new valid tokens T_a and T_b corresponding to the requested outputs.

The operation supports the special cases $T_2 = \perp$ and $(P_b, v_b) = \perp$.

Non interactive Pay

This operation allows a user to transfer value to user u without interaction from u, by increasing the un minted balance of u.

A user submits two valid unspent tokens T_1, T_2, together with a transaction request specifying the recipient (pk_u, v) and information for creating a change token (P_b, v_b). The servers verify that $v_1 + v_2 = v + v_b$ and that the whole request is properly authorized by signatures under the public keys associated with T_1 and T_2. If so, the servers atomically mark T_1 and T_2 as spent, issue a new valid change token T_b, and increase the un minted balance of user u by v.

Prove that your scheme satisfies the following properties.

1. **Unforgeability.** No adversary can create a valid token unless it is issued by the servers as part of a successful Mint or Pay operation.

2. **Authorization Unforgeability.** No adversary can cause a token to be spent without producing a valid authorization signature under the public key embedded in that token.

3. **No Double Spend.** No valid token can be successfully spent more than once.

4. **Liveness.** If an honest user holds a valid unspent token and initiates a Mint or Pay operation, then the operation completes successfully in a constant number of rounds.

5. **Conservation of Value.** At all times, the total value represented by all valid unspent tokens and all un minted balances in the system remains invariant.

2 Verifiable Secret Sharing

Consider a Verifiable Secret Sharing (VSS) protocol for $n = 2f + 1$ parties with omission failures only. There is a designated *dealer* holding an input value s, and two protocols: *share* and *recover*.

Remark. In the omission only failure model, polynomial degree violations cannot occur due to adversarial behavior. The main challenge is ensuring that all honest parties receive a share.

Termination

- If all non faulty parties invoke *share*, then all non faulty parties terminate within a constant number of rounds.
- If all non faulty parties invoke *recover* after terminating *share*, then all non faulty parties terminate within a constant number of rounds.

Hiding

- If the dealer is non faulty, then for any set F of at most f parties, there exists an efficient simulator $\mathcal{S}$ that produces a distribution indistinguishable from the joint view of the adversary during *share*, independently of s.

Validity

- If the dealer is non faulty and *recover* is invoked after *share* terminates, then the output of *recover* equals the dealer input s.

Binding

- If the dealer is faulty, then there exists an efficient extractor $\mathcal{E}$ that, given the views of all non faulty parties at termination of *share*, outputs a value b such that any subsequent invocation of *recover* outputs b.

Assume that in addition to sending a plaintext message $< m >$ to party j, a sender may transmit a message inside an envelope $< E_j(m) >$ that only party j can open. Other parties may forward the envelope but cannot

inspect its contents. This abstraction captures an ideal public key encryption primitive with confidentiality only.

Prove that a VSS protocol satisfying the above properties exists for $n = 2f + 1$. Analyze its round complexity and message complexity.

3 Distributed Key Generation and Agreement

Design a Distributed Key Generation (DKG) protocol such that, at termination, all non faulty parties hold shares of a uniformly random polynomial g of degree at most f and the servers output a public key pk that corrisponds to the secret key $g(0)$.

In particular, assume there exists a knonw group element h and that the public key is $h^g(0)$.

Prove termination and show that all non faulty parties terminate within one round of each other. Your construction may use a consensus protocol or a broadcast protocol and the VSS protocol defined above. Clearly specify the properties of any broadcast or agreement protocol you rely on.

Prove hiding: that conditioned on any view of f parties, the secret key is uniformly randomly distributed.

Prove validity and agreement: all parties see the same pubic key and that this public key corrisponds to the secret key inducded by the signatures shares of the parties.

Prove liveness: all non-fualty servers end up with a signature share.

Analyze the round complexity and message complexity of your solution.

4 Bonus

Extend your constructions to the Byzantine failure model. You may assume a PKI providing both digital signatures and public key encryption.