



# PE TM Chain Hash Generation

User Manual  
For Enterprises & Telemarketers



## Overview

This document outlines the necessary steps required for TM-Delivery Function to generate the hash basis the Chain defined on the DLT portal.

It is expected to generate the hash using SHA256.

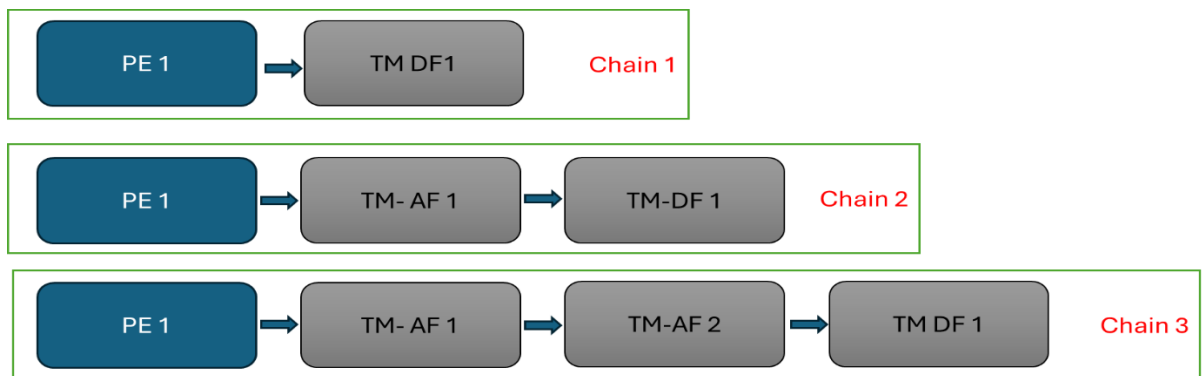
Below steps for Hash Generation by TM-DF will be applied after the chain is approved at all stages by all respective stakeholders like TM-AF as well as TM-DF. The Selection of TM-DF will mark the end of chain and it will be assumed that traffic will be submitted by TM-DF to the Telco DLT Network. Once the TM-DF is marked in the chain and its accepted by the TM-DF, a final request will go to PE for the approval of end to end chain.

TM-DF are required to generate hash and submit the same in the TLV parameter, the process for which will be shared separately with all TM-DF.

All the stakeholders like PE, TM-AF and TM-DF will append their DLT ID in the live SMS traffic and TM-DF will be responsible to collate all ID's and generate hash before submitting to Telco DLT Network.

## Hash Generation

Below illustration shows three different chains initiated by PE and approved on DLT after acceptance from respective TM-AF or TM-DF as applicable.



Following sample data is assumed as a illustration for Hash generation.

**PE ID:** 1101172974420810001 ;

**TM AF1 ID:** 1101172974420820002 ;

**TM AF2 ID:** 1101172974420820003 ;

**TM DF1 ID:** 1101172974420830004

## Possible Chains and hash

### Chain 1: PE to TMDF1,

Hash will be SHA256(PEID,TMDF1ID)

### Chain 2: PE to TMAF1 to TMDF1,

Hash will be SHA256(PEID,TMAF1ID,TMDF1ID)

### Chain 3: PE to TMAF1 to TMAF2 to TMDF1

Hash will be SHA256(PEID,TMAF1ID,TMAF2ID,TMDF1ID)

Hash generation logic will be SHA256 starting with PE and ending with TMDF with comma as a separator in between with no space.

## Program Example (Golang)

-----

```
package main
```

```
import (  
    "crypto/sha256"  
    "encoding/hex"  
    "fmt"  
)
```

```
func GenerateSHA256Signature(input string) string {
```

```
    hasher := sha256.New()  
    hasher.Write([]byte(input))  
    hexHash := hex.EncodeToString(hasher.Sum(nil))  
    return hexHash  
}
```

```
func main() {
```

```
    chain1 :=  
    "1101172974420810001,1101172974420830004"  
    hashofpedf := GenerateSHA256Signature(chain1)  
    fmt.Println("Chain1: ", hashofpedf)
```

```
    chain2 := "1101172974420810001,1101172974420820002,1101172974420830004"  
    hashofpetmdf := GenerateSHA256Signature(chain2)  
    fmt.Println("Chain2: ", hashofpetmdf)
```

```
chain3 :=
"1101172974420810001,1101172974420820002,1101172974420820003,11011729744
20830004"
    hashofpetmtmdf := GenerateSHA256Signature(chain3)
    fmt.Println("Chain3: ", hashofpetmtmdf)
}
```

## Output

### Chain1

c0c951afca257c5b87c3da535c09112080483f35b7e6faf35e2428b3b526b694

### Chain 2

ba38bbe4ab3780a70aad6071c006e720eed8efc1ff70b00691d3b6844e6497e2

### Chain3

d3133e436040034d9e95932ea93a1f82c37e1d98803229f835d8f3124e77e07f

## Program Example (Java)

-----

```
import java.util.*;
import java.security.MessageDigest;

class CreateChainHash {
    public static void main(String[] args) {
        String chain1 = "1101172974420810001,1101172974420830004";
        String hashofpedf = getHash(chain1);
        System.out.println("hashofpedf: "+ hashofpedf);
        String chain2 =
"1101172974420810001,1101172974420820002,1101172974420830004";
        String hashofpetmdf = getHash(chain2);
        System.out.println("hashofpetmdf: "+ hashofpetmdf);
        String chain3 =
"1101172974420810001,1101172974420820002,1101172974420820003,11011729744
20830004";
        String hashofpetmtmdf = getHash(chain3);
        System.out.println("hashofpetmtmdf: "+ hashofpetmtmdf);
    }

    static String getHash(String input) {
        String hash = "";
        try {
            MessageDigest digest = MessageDigest.getInstance("SHA-256");
            byte[] valbyte = digest.digest(input.getBytes());
```

```
StringBuilder hexString = new StringBuilder(2 * valbyte.length);
for (byte b : valbyte) {
    String hex = Integer.toHexString(0xff & b);
    if (hex.length() == 1) {
        hexString.append('0');
    }
    hexString.append(hex);
}
hash = hexString.toString();
}
catch (Exception e) {
    System.out.println("error" + e);
}
return hash;
}
}
```

### Output:

#### Chain 1

c0c951afca257c5b87c3da535c09112080483f35b7e6faf35e2428b3b526b694

#### Chain2

ba38bbe4ab3780a70aad6071c006e720eed8efc1ff70b00691d3b6844e6497e2

#### Chain3

d3133e436040034d9e95932ea93a1f82c37e1d98803229f835d8f3124e77e07f

### TLV Parameter

Following TLV tagid will be used for publishing the generated hash.

TLV TagID: 5122

### SMPP Error Codes

Following error codes will be generated from BSNL DLT for various permutations and combinations of errors on generated Hash.

ErrorCodes:

PE\_TM\_HASH\_NOT\_RECEIVED = 614

PE\_TM\_HASH\_NOT\_REGISTERED = 615

PE\_TM\_HASH\_INACTIVE = 616

PE\_TM\_HASH\_BLACKLISTED = 617

PE\_TM\_HASH\_SUSPENDED = 618