tangem | GIMLY

Made for blockchain NFC: manual

Powered by Tangem, provided by Gimly

Gimly
Kraanspoor 7E2
1033 SC, Amsterdam

T: +31 (0)20 786 6452
E: caspar@gimly.io
W: www.gimly.io

KvK: 75570750
BTW: NL002189334B65
IBAN: NL79 KNAB 0259516260

Contents

1. Gimly: Tangem implementation partner

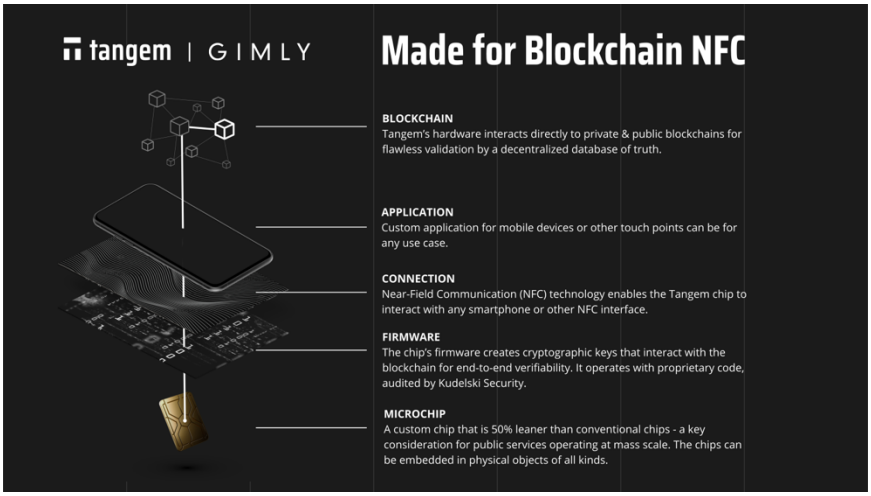
Gimly is official Tangem supplier and integration partner. Gimly helps clients and partners identify and realise the added value of made-for-blockchain NFC and IoT, by:

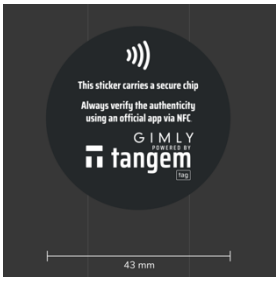
1. End-to-end solutions development:
    - Use case and product definition, and full development services.
2. Technology partnering:
    - Joint use case and product definition;
    - Supply of pre-programmed NFC tags, stickers, or cards: low volumes for testing and piloting;
    - Supply of fully programmable NFC developer cards: low volumes for testing and piloting;
    - High volume supply of personalised made-for-blockchain NFC tags, stickers, or cards, with customised settings and design: high volumes for operational environments.

Get in touch with Gimly to start building with Tangem here: https://www.gimly.io/tangem-by-gimly.

2. Made-for-blockchain NFC summary

With Tangem made-for-blockchain NFC chips, physical objects can interact securely and directly with digital blockchains. No third layer, cloud service, or back-end needed for key management and signature generation. The private key is securely generated within the chip, cannot be extracted, and is used to sign blockchain transactions directly from the chip. *Contrary to other crypto tags, Tangem made-for-blockchain NFC chips do not only allow for authentication through challenge response schemes, but they can sign multiple types of transactions directly from the chip.* They offer a wide range of additional security, 2fa and data security functions, and are available as physical cards, tags or stickers. With their different form factors and broad functionality, they open up a wide range of use cases, including anti-counterfeiting, supply chain and logistics, and identity solutions.





- Sign blockchain transactions: raw/hashed/single/arrays of transactions directly from the chip.
- Support sec256k1 and ed25519 cryptographic curves.
- Authenticity: prove product and chip authenticity through challenge response.
- Data storage: sign and store encrypted or raw data on chip.
- 2FA: allow only transactions or data validated through issuer signature.
- Other security functions: Security delays, PIN codes, CVC codes.
- EAL6+ certified secure smart-card chip based on ARM SC000 core architecture and having ISO14443 Type A contactless interface.
- SSI cards: support for decentralized identifyers (DIDs) and Verifyable Credentials (VCs) for self-sovereign identity (SSI) solutions. See Appendix E: WRITE_FILEDATA for writing private files (supported with firmware version 3.29+).
- SDKs for integrating Tangem technology in your solutions here: https://github.com/Gimly-Blockchain/Tangem-Blockchain-NFC.

The Tangem card is a self-custodial hardware wallet for blockchain assets. The main functions of Tangem cards are to securely create and store a private key from a blockchain wallet and sign blockchain transactions. Tangem card does not allow users to import/export, backup/restore private

keys, in this way guarantying that the wallet is unique and unclonable. Above this, the cards provide mechanisms to implement two-factor authentication of transactions, off-line validation of wallet balance, protection against counterfeit and cloning, secure signing of transactions on POS devices, and more. All these capabilities are described in this document.

The cards carry an EAL6+ certified secure smart-card chip based on ARM SC000 core architecture and having ISO14443 Type A contactless interface. Tangem's card firmware is a native card OS (**COS**) providing operations with a blockchain wallet and a proprietary communication protocol on top of ISO14443 to interact with a contactless terminal (*host*). Once loaded into a particular card, COS binary code cannot be updated or managed.

The host application (**App**), which is almost always installed on an NFC smartphone, provides UI and interaction with the card via Android or iOS NFC interface. Tangem NFC protocol is not compatible with older Android smartphones having NFC modules that do not support long (> 261 bytes) APDUs. Due to the fact that old iOS versions support only read-only NDEFs, functionality on old iOS is limited to the reading of the card and non-strict balance validation by using a mechanism described in section Dynamic NDEF. Beginning from iOS 13, all NFC-enabled iPhone devices support all features of the Tangem NFC protocol.

2.1 General life cycle

In the initial state, right after the COS is loaded, the card's Status is set to 'New'. In this status, the card will only accept PERSONALIZE command from the host. This command can be executed only once to define the principal parameters of the card. Host must pass all parameters AES256-encrypted with Personalization_Key in one request payload. Once the command is executed, these parameters become read-only, and the card is rendered to the initial state of the main cycle with Status set to 'Empty'. At this moment, the card is ready for operation by the end-user and contains all data except for the blockchain wallet keys.

App must request COS to generate a wallet key pair by calling CREATE_WALLET command. This command will render the card to 'Loaded' state and return wallet's public key to the user, who can top up the newly created wallet in the blockchain using any third-party wallet app. In a default configuration, COS is not aware of incoming transactions and wallet balance, although COS provides two optional mechanisms to store trusted balance on-card and enable offline balance verification (described below). COS in 'Loaded' state supports all commands, including SIGN. Calling SIGN command will make COS generate and return a transaction signature.

If the card depletes the number of allowed signatures defined during personalization, or if the user opts to delete the wallet by calling PURGE_WALLET command, COS will destroy all wallet data and switch either to 'Purged' state, or back to 'Empty' state, depending on the reusability option that is also defined during personalization. 'Purged' state is final, it makes the card useless.

2.2 NFC communication protocol:

NFC protocol is adapted from ISO14443-3 layer, not fully compliant with ISO14443-4. It has a proprietary set of features tailored to blockchain usage. The protocol contains only atomic commands, meaning that every command makes some complete action, and either is fully executed or fully rolls back to the previous card state.

- App should obtain 4-byte card UID code when initially establishing communication via ISO14443-3 stack.
- Standard ISO7816-4 format is used for all requests: [CLA, INS, P1, P2, Lc, Payload]. Maximum length of command is limited to 1024 bytes.
- Payload in TLV format <Tag, Length, Value>, where:
- Tag - 1 byte - field ID,
- Length - 1 or 3 bytes - value length
- Value - [Length] byes - field value.
- Field value format:
- The most significant byte value is at the lowest address (Big-endian)
- Strings can be null-terminated in utf-8 encoding
- Public key - as uncompressed (0x04, X[32], Y[32]) for 'secp256k1' curve or as compressed X[32] for 'ed25519' curve
- Signature - as (R[32], S[32])
- Example, Read_Card Request :

Request

>> 00F200002210291B4D142823F7D20C5F08DF69122DE43F35F057A988D9619F6D3138485C9A203

CLA	INS	P1	P2	Lc	Payload		
					Tag	Length	Value
00	F2	00	00	22	10 (PIN1)	20	91 B4 D1 42 82 3F 7D 20 C5 F0 8D F6 91 22 DE 43 F3 5F 05 7A 98 8D 96 19 F6 D3 13 84 85 C9 A2 03

- COS supports three options of communication. App can select encryption mode by setting the corresponding P2 parameter in a command request. If the card does not support chosen encryption mode, it will respond with error status word SW*NEED*ENCRYPTION = 0x6982

Encryption modes:

1. Plain data packets with non-encrypted Payload: This option is not recommended for large-scale commercial projects, because it potentially exposes the card to eavesdropping and replay attacks.
2. Fast symmetric encryption with mutual challenges: If non-default PIN1 is set, the communication is encrypted by AES256 and safely protected against man-in-the-middle and replay attacks. Packets could be potentially eavesdropped and decrypted later by brute-forcing weaker PIN1 codes. However, this attack has little practical meaning. Fast encryption is optimal for most use cases.

3. Strong encryption (default mode) with ECDH shared secret (elliptic curve Diffie-Hellman): Communication is encrypted by AES256 and fully protected against eavesdropping, man-in-the-middle and replay attacks. The downside is 2x lower performance and longer card's reaction time.

2.3 Card commands and personalisation options:

Note: INS for personalisation command not currently available. However, Gimly provides the Tangem developer app for personalization with every developer kit.

Software should be able to call all COS commands, plus Personalize command:

1. Personalize
2. ACTIVATE_CARD (INS code: 0xFE)
3. READ_CARD: (INS code: 0xF2)
4. CREATE_WALLET : (INS code: 0xF8)
5. CHECK_WALLET : (INS code: 0xF9)
6. SET_PIN : (INS code: 0xFA)
7. SIGN
    1. Wallet mode (INS code: 0xFB)
    2. POS mode (INS code: 0xFB)
8. READ_ISSUER_DATA (INS code: 0xFB)
    1. Read issuer extra data (INS code: 0xF7)
9. WRITE_ISSUER_DATA (INS code: 0xF6)
    1. Write issuer extra data (INS code: 0xF6)
10. VERIFY_CODE (INS code: 0xF5)
11. VERIFY_CARD (INS code: 0xF3)
12. VALIDATE_CARD (INS code: 0xF4)
13. PURGE_WALLET (INS code: 0xFC)
14. READ_USER_DATA (INS code: 0xE1)
15. WRITE_USER_DATA (INS code: 0xE0)
16. (NEW) WRITE_PRIVATEFILE

3. Security

3.1 General

Tangem card guarantees that it is the only place in the world that holds the private key of the blockchain wallet (address). Therefore, it is not possible to export, import, backup, restore, derive or in some other way gain access to the wallet private key. Loss or physical destruction of the card is equivalent to loss of the wallet's funds.

Tangem cards pass rigorous testing and can withstand environmental extremes and occasional mechanical deformation within limits defined in ISO7810 standard. It is recommended to avoid intentional bending with force and exposure to temperatures above 80C, powerful X-rays and magnetic field (e.g. MRI). The cards' microprocessor employs many anti-tampering mechanisms that can recognize various types of attacks. Tangem COS will react to attempts of such attacks according to its severity. In some situations, the card will permanently erase all wallet data in order to prevent unauthorized access to the private key. Cards will also withstand unpredicted power outage that may occur when NFC field or the host device is removed from the card. On-card data integrity is protected by a proprietary anti-tearing mechanism and triple storage redundancy. Embedded non-volatile memory (NVM) is certified and tested for 20 years of the data retention period.

3.2 Card attestation

In the process of manufacturing, every new Tangem card internally generates a Card Key pair *Card_PublicKey / Card_PrivateKey*. The private key *Card_PrivateKey* is permanently stored in the card memory and is not accessible to external applications via NFC interface. At the same time, Tangem publishes the list of *CID* and corresponding *Card_PublicKey* values in its card attestation service and/or hands over this list to the card Issuer.

To attest the card, app should:

1. call READ_CARD command to obtain CID and Card_PublicKey
2. call VERIFY_CARD command to ensure the card has not been counterfeited: by using the standard challenge-response scheme, the card proves possession of Card_PrivateKey that corresponds to Card_PublicKey returned by READ_CARD command,
3. ensure that the card's *CID* presents and corresponds to *Card_PublicKey* in published Tangem's card list, through either Tangem attestation service or Issuer's own service,
4. optionally, in offline, verify *Manufacturer_Signature* using non-secret *Manufacturer_PublicKey* stored in the app.

3.3 Issuers

Issuer is a third-party team or company wishing to use Tangem cards in their project or business. Issuers would have their own distribution channel and can tailor Tangem solution to their needs:

- to choose a type of blockchain assets Tangem card carries (e.g. ERC20 tokens),

- to integrate Tangem cards into own mobile host application connected to issuer's own server back-end or some third-party nodes/services,
- to order customized cards from Tangem: it could be a new graphic design and personalization parameters that depend on the issuer's use-case and security model.
- Tangem will remain responsible only for the card's security and card attestation process.

3.3.1 Issuer transaction key

Tangem cards support the optional security mechanism allowing the issuer to authorize transactions before they are signed by the user. In case the issuer opts to utilize this mechanism, the issuer has to generate Issuer Transaction Key pair *Issuer_Transaction_PublicKey / Issuer_Transaction_PrivateKey* either for each card or for the whole batch. The private key *Issuer_Transaction_PrivateKey* is permanently stored in a secure back-end of the issuer (e.g. HSM). The public key *Issuer_Transaction_PublicKey* is securely written into the Tangem card during personalization by Tangem and will be permanently stored there throughout the whole card life cycle, with no access by host application via NFC interface.

App shall call READ_CARD command to get *Signing_Method* parameter that defines what data should be submitted to SIGN command. If *Signing_Method* is '2', '3', '4', or '5', then the card will sign only those data previously signed by the issuer. App has to obtain issuer's signature of the data from the issuer's own back-end services before submitting it to SIGN command. This mechanism enables two-factor authentication of transactions by using external issuer services.

3.3.2 Issuer data key

Issuer may also use a special 512-byte memory block *Issuer_Data* to securely store and update information in COS. For example, this mechanism could be employed for enabling off-line validation of the wallet balance and attesting of cards by the issuer (in addition to Tangem's attestation). The issuer should define the purpose of use, payload, and format of *Issuer_Data* field. Note that *Issuer_Data* is never changed or parsed by the executable code the Tangem COS.

The issuer has to generate single Issuer Data Key pair *Issuer_Data_PublicKey* / *Issuer_Data_PrivateKey*, same for all issuer's cards. The private key *Issuer_Data_PrivateKey* is permanently stored in a secure back-end of the issuer (e.g. HSM). The non-secret public key *Issuer_Data_PublicKey* is stored both in COS (during personalization) and issuer's host application that will use it to validate *Issuer_Data* field received from READ_ISSUER_DATA command.

In case the issuer opts to utilize *Issuer_Data* field, there are two options:

1. *Issuer_Data* is written into COS by host application by using WRITE_ISSUER_DATA command. This option works only if Signing_Method is set is '0', '1', '2', or '3'. *Issuer_Data* can be either permanent and defined during personalization, or updatable during the life cycle. Updating of *Issuer_Data* should not be bound to a specific transaction and should tolerate unpredictable delays caused by the end-user not tapping the card for a long period of time. The issuer's host

application should obtain the signature *Issuer_Data_Signature* of *Issuer_Data* from issuer's own back-end services. This is not a part of the solution provided by Tangem team.

2. *Issuer_Data* is updated strictly during the execution of SIGN command. This option works only if Signing_Method is set to '4' or '5'. For this option, WRITE_ISSUER_DATA will not work and will return error with status word SW_ERROR_PROCESSING_COMMAND = 0x6286

Version 2.30 and later.

Issuer may also use extended 32kB memory block *Issuer_Extra_Data* to securely store and update information within the card. For example, this mechanism could be employed for storing personal biometric data on ID cards. This memory can be marked as "write once" by specifying *RestrictOverwriteIssuerDataEx* flag in *SettingsMask*.

3.4 Acquirer

Version 2.30 and later.

Acquirer is a trusted third-party company that operates proprietary (non-EMV) POS terminal infrastructure and transaction processing back-end. Tangem cards personalized with the acquirer key have a different flow of signing transactions received from host devices (POS terminals) authorized by the acquirer. Most importantly, COS will not request PIN2 and will not require security delay protection if SIGN command is called by such a trusted POS terminal. This way Tangem cards provide secure instant single-tap payments on POS terminals.

3.4.1 Acquirer key

Optional acquirer public key can be loaded onto card during personalization and used to authorize acquirer's terminal. Acquirer signs a certificate of such terminal with the acquirer's private key, then securely uploads it into each POS terminal. If enabled by a flag in *SettingsMask*, the card will verify POS terminal certificate signature with the acquirer's public key before signing a POS transaction.

3.4.2 Card shared secret key

64-bytes secret key stored during the personalization process on both card and acquirer processing server and used as a seed to encrypt POS transaction by COS and decrypt transaction by acquirer processing server.

3.5 Wallet key

App shall call CREATE_WALLET command to turn an empty Tangem card into a 'cold' blockchain wallet ready for top-up. This command will generate a wallet key pair *Wallet_PublicKey / Wallet_PrivateKey*.

The private key *Wallet_PrivateKey* is the main secret of the card. It is never revealed and accessible by host application via NFC interface. COS will internally use *Wallet_PrivateKey* only during execution of SIGN and CHECK_WALLET commands.

App will need to obtain *Wallet_PublicKey* from the response of CREATE_WALLET or READ_CARD commands and then transform it into an address of corresponding blockchain wallet according to a specific blockchain algorithm.

Calling of PURGE_WALLET command will permanently delete the wallet key pair. If the card was personalized as reusable (see *Is_Reusable* flag in *Settings_Mask*) then App may call CREATE_WALLET command to generate a new wallet key pair.

3.6 User's codes

COS provides three codes to protect the card: PIN1, PIN2, CVC. The application should submit one, two, or all of these codes in command parameters in order to be authorized by COS.

The scheme of protection and personalization should be defined by the issuer as follows:

- what codes are required,
- how codes are used or changed by end-users,
- how codes are delivered to end-users,
- initial values of the codes set during personalization.

The card protects itself against brute force snigging of PIN1, PIN2, CVC codes. After each invalid code is submitted, COS gradually increases the delay of response to all further commands until the correct code is submitted. Therefore, the card will never block itself regardless of the number of invalid inputs. However, when the correct code if submitted after a few invalid codes, the user will have to await card's response for the amount of time proportional to the number of prior invalid inputs. The response delay will be set to zero only after the card receives the correct code.

Version 1.19 and later.

The delay of PIN1 response will not increase in case NFC encryption is disabled and host submits default PIN1. PIN2 response delay will also not increase when host submits default PIN2. It this way, the host can verify that the card has default PIN values without future delay penalty.

Version 2.30 and later.

COS provides additional PIN3 to protect the card in POS transaction process.

3.6.1 PIN1 code

This 32-byte code restricts access to the whole card. App must submit the correct value of PIN1 in each command, including READ_CARD (at the moment, when CID is not yet known to the app). By default, all cards have PIN1 set to SHA256('000000'), so that every host application on any NFC smartphone could obtain card and wallet data. The end-user might optionally restrict access to the whole card and all commands by setting a user's PIN1 code. Issuer can disable changing PIN1 by defining a special parameter during personalization (see *Allow_SET_PIN1* flag in *Settings_Mask*).

A non-default PIN1 is required for remote card activation (in this case PIN1 is used as a card activation key). If the card is used as a hardware wallet for storing high-value assets, it is also recommended to require the end-user to set a non-default PIN1 at least eight characters long.

For additional security purposes, COS stores all wallet data, including *Wallet_PrivateKey*, in NVM encrypted with AES256 key derived from PIN1.

3.6.2 PIN2 code

All cards will require submitting the correct 32-byte PIN2 code in order to sign a transaction or to perform other commands entailing a change of the card state. App should ask the user to enter PIN2 before sending such commands to the card. The main purpose of using PIN2 is to protect against proximity attack, during which an attacker tries to discreetly scan / detect Tangem cards with default PIN1 in proximity and withdraw assets from such cards by signing a transaction.

Issuer can disable changing PIN2 by defining a special parameter during personalization (*Allow_SET_PIN2* flag in *Settings_Mask*). The recommended minimum length of user's PIN2 is six characters.

By default, all cards have PIN2 set to SHA256('000'). Default PIN2 should not be used if *CVC* and *Pause_Before_PIN2* are not used. If the card is being handed over between from one holder to another, PIN2 should be temporarily set to a default value SHA256('000'), so that the receiving holder could validate and change PIN1 and PIN2 codes by calling SET_PIN command. Once the new hold has the card in possession, he/she should immediately set own secret PIN2.

3.6.3 PIN3 code

Version 2.30 and later.

When POS terminal request card to sign POS transaction and transaction amount exceed PIN3 floor limit (PIN3_Floor_Limit) card additionally encrypt transaction signature with key derived from PIN3. In this case, POS terminal shall request the user to enter PIN3 to confirm and decrypt POS transaction so it can be broadcasted to blockchain.

See "SIGN – POS mode" section. See flow diagram in "Tangem Payment Flow v2.1.pdf" document.

3.6.4 CVC code

Issuer can require using a CVC code with non-transferrable cards (e.g. a personal reusable cold wallet or one-off card). CVC code will be loaded into COS and printed on the card during personalization by Tangem. CVC value is fixed during the whole card life cycle. If it is set, the card will require submitting correct CVC code in order to sign a transaction or to perform other commands entailing a change of the card state. App should verify if CVC is enabled (*Use_CVC* flag in *Settings_Mask*) and then ask the user to enter CVC before sending such commands to the card.

The purpose of using CVC is the same as for PIN2 – to protect the card against 'drive by' attack. However, it allows keeping it more simple for the user. If PIN1 and PIN2 always stay default and never asked in application UI, the user needs only to read and enter CVC printed on the card.

3.7 Linked terminal

Version 2.30 and later.

App can optionally generate ECDSA key pair *Terminal_PrivateKey / Terminal_PublicKey.* And then submit *Terminal_PublicKey* to the card in any SIGN command. Once SIGN is successfully executed by COS, including PIN2 verification and/or completion of security delay, the submitted *Terminal_PublicKey* key is stored by COS. After that, the App instance is deemed trusted by COS and COS will allow skipping security delay for subsequent SIGN operations thus improving convenience without sacrificing security.

In order to skip security delay, App should use *Terminal_PrivateKey* to compute the signature of the data being submitted to SIGN command for signing and transmit this signature in *Terminal_Transaction_Signature* parameter in the same SIGN command. COS will verify the correctness of *Terminal_Transaction_Signature* using previously-stored *Terminal_PublicKey* and, if correct, will skip security delay for the current SIGN operation.

This behavior can be enabled by setting flag *Skip_Security_Delay_If_Validated_By_Linked_Terminal* in *Security_Mask*.

3.8 Security Delay

COS provides a special mechanism to protect against proximity attack on a card with default or very weak PIN1 and PIN2. For that, COS can enforce a "security delay" up to 60 seconds long between reception and execution of a command. During the delay, the card shall be kept within NFC field of the terminal and NFC session shall be active until the delay countdown is finished on the card.

See details in section "Pending security delay".

There are a few flags in *Setting_Mask* field to control security delay behavior:

- *Pause_Before_PIN2* enforces and defines the amount of the delay before COS executes any of the commands protected by PIN2. It is recommended to apply at least 10 seconds delay if the card assumes using the default PIN2. Pause_Before_PIN2 is defined during personalization.
- *Smart_Security_Delay* flag in *Settings_Mask* will make COS automatically disable the security delay if non-default PIN2 is set on the card.

Version 2.05 and later

- *Skip_Security_Delay_If_Validated_By_Issuer* will skip security delay in SIGN command if the issuer validates the transaction (for signing methods 2, 3, 4 and 5).

Version 2.30 and later

- *Skip_Security_Delay_If_Validated_By_Linked_Terminal* will skip security delay in SIGN command if the transaction was signed by linked terminal (for signing methods 0, 1).

4. NFC communication

4.1 General

Tangem NFC protocol is based on ISO14443-3 layer, but do not fully comply with ISO14443-4. It has a proprietary set of features tailored to blockchain usage. The protocol contains only atomic commands, meaning that every command makes some complete action, and either is fully executed or fully rolls back to the previous card state.

App should obtain 4-byte card UID code when initially establishing communication via ISO14443-3 stack.

Standard ISO7816-4 format is used for all requests: [CLA, INS, P1, P2, Lc, Payload]. Maximum length of command is limited to 1024 bytes.

COS supports three options of communication:

1. Plain data packets with non-encrypted Payload:

This option is not recommended for large-scale commercial projects, because it potentially exposes the card to eavesdropping and replay attacks.

2. Fast symmetric encryption with mutual challenges:

Version 1.16 and later.

If non-default PIN1 is set, the communication is encrypted by AES256 and safely protected against man-in-the-middle and replay attacks. Packets could be potentially eavesdropped and decrypted later by brute-forcing weaker PIN1 codes. However, this attack has little practical meaning. Fast encryption is optimal for most use cases.

3. Strong encryption with ECDH shared secret (elliptic curve Diffie-Hellman):

Communication is encrypted by AES256 and fully protected against eavesdropping, man-in-the-middle and replay attacks. The downside is 2x lower performance and longer card's reaction time.

Every Tangem card supports strong encryption mode. Issuer can opt to enable fast encryption and non-encrypted mode during personalization. App can choose desirable encryption mode among those supported by the card by setting the corresponding P2 parameter in a command request as described below. If the card does not support chosen encryption mode, it will respond with error status word SW_NEED_ENCRYPTION = 0x6982.

4.2 TLV format (SimpleTLV)

TLV structure is a list of triples <Tag, Length, Value>, where:

- Tag - 1 byte - field ID,

- Length - 1 or 3 bytes - value length,
- Value - [Length] byes - field value.

```
TLV1
      TAG1: 1 byte
      Length1: 1 byte (length 1–255) or 3 bytes with extended coding [0xFF, high, low]
      Value1: [Length1 bytes]
TLV2
...
TLVm
```

Field value format:

- The most significant byte value is at the lowest address (Big-endian)
- Strings can be null-terminated in utf-8 encoding
- Public key - as uncompressed (0x04, X[32], Y[32]) for ‘secp256k1’ curve or as compressed X[32] for ‘ed25519’ curve
- Signature - as (R[32], S[32])

4.3 Non-encrypted request

Tangem COS non-encrypted requests:

- CLA, P1, P2 = 0x00
- INS – command code
  - 1 byte (e.g. READ_CARD = 0xF2, see Commands section)
- Lc – length of Payload
  - 1 byte (length 1-255) or 3 bytes with extended coding [0x00, high, low]
- Payload – command parameters in TLV format

Tangem COS non-encrypted response:

- Response_Data – command response in TLV format
- Status_Word – command result (2 bytes, bigendian)

Status words used:

Status	**Bytes**
SW_PROCESS_COMPLETED	0x9000
SW_NEED_ENCRYPTION	0x6982
SW_INVALID_PARAMS	0x6A86

SW_ERROR_PROCESSING_COMMAND	0x6286
SW_INVALID_STATE	0x6985
SW_PINS_NOT_CHANGED	0x9000
SW_PIN1_CHANGED	0x9001
SW_PIN2_CHANGED	0x9002
SW_PINS_CHANGED	0x9003

Version 2.30 and later.

SW_PIN3_CHANGED	0x9004
SW_PINS_CHANGED	0x900X, X – changed PINs mask (0x01-PIN1, 0x02-PIN2, 0x04-PIN3)
SW_EXT_SIGN_BLOCKED	0x6983

Data types and structures:

- String values have maximum size of 16 bytes, 0x00 is the end symbol
- Public key – 65 bytes, elliptic curve point P(X,Y) in uncompressed format
    - byte[0]=0x04
    - byte[1..32] - X coordinate,
    - byte[32..64] - Y coordinate
- Signature – 64 bytes [R[32],S[32]]

4.4 Fast encrypted request

Tangem COS fast encrypted request:

- CLA, P1 = 0x01, P2 = 0x00
- INS – command code - 1 byte
- Lc – length of Payload - 1 byte (length 1-255) or 3 bytes with extended coding [0x00, high, low]
- Payload:

1) Prepare tlv_data array by concatenating:
    a. Length of TLV data (as per non-encrypted request format, see above) - 2 bytes
    b. CRC16 of TLV data - 2 bytes

c. TLV data

2) Generate random 16-byte array [challengeA] in App, call OPEN_SESSION command with P2 = 0x01 and challengeA as a parameter,
3) COS will generate and return random 16-byte array [challengeB] as a response to OPEN_SESSION command,
4) calculate protocol_key = PBKDF2(SHA256(PIN1), UID, 50), where UID - ISO 14443-3 unique identifier,
5) calculate session_key = SHA256(challengeA | challengeB | protocol_key),
6) Payload = AES256(session_key, tlv_data).

Tangem COS fast encrypted response:

- Response_Data – contains concatenated:
  - Length of response TLV data - 2 bytes
  - CRC16 of response TLV data - 2 bytes
  - AES256(session_key , response_tlv_data)
- Status_Word – command result (2 bytes, big-endian)

CRC16 of TLV data should always be verified both by App and COS to exclude dealing with corrupted or tampered data. CRC16 in Tangem means CRC-A implementation of ISO 14443-3. See Java sample code in the Appendix B.

4.5 Strong encrypted request

Tangem COS strong encrypted request:

- CLA, P1 = 0x02, P2 = 0x00
- INS – command code - 1 byte
- Lc – length of Payload -
  - 1 byte (length 1-255) or 3 bytes with extended coding [0x00, high, low]
- Payload –

1) Prepare tlv_data array by concatenating:

- Length of TLV data (as per non-encrypted request format, see above) - 2 bytes

- CRC16 of TLV data - 2 bytes

- TLV data

2) Generate a key pair [privA, pubA] using secp256k1 curve in App,
3) Call OPEN_SESSION command with P2 = 0x02 and [pubA] key as a parameter,
4) COS will generate own a key pair [privB, pubB] and return [pubB] as a response to OPEN_SESSION command,
5) Calculate shared_secret = privA * pubB as per ECDH protocol,
6) Calculate protocol_key = PBKDF2(SHA256(PIN1), UID, 50), where UID - ISO 14443-3 unique identifier,
7) Calculate session_key = SHA256(shared_secret | protocol_key),

8) Payload = AES256(session_key, tlv_data).

Tangem COS strong encrypted response:

- Response_Data – contains concatenated:
  - Length of response TLV data - 2 bytes
  - CRC16 of response TLV data - 2 bytes
  - AES256(session_key , response_tlv_data),
- Status_Word – command result (2 bytes, bigendian)

CRC16 of TLV data should always be verified both by App and COS to exclude dealing with corrupted or tampered data.

4.6 OPEN_SESSION command

In case of encrypted communication, App should setup a session before calling any further command. OPEN_SESSION command generates secret session_key that is used by both host and card to encrypt and decrypt commands' payload (see above).

Command INS code: **0xFF**

4.6.1 Fast Encryption

P1=0x00

P2=0x01

Parameters:

Field	Tag	Type	Description
ChallengeA	0x1A	byte[16]	Host's challenge

Response:

Field	Tag	Type	Description
ChallengeB	0x1B	byte[16]	Card's challenge

UID	0x0B	byte[4]	*Version 2.30 and later.* ISO 14443-3 unique identifier – for compatibility with devices where UID is hidden

4.6.2 Strong Encryption

P1=0x00

P2=0x02

Parameters:

Field	Tag	Type	Description
pubA	0x1A	byte[65]	Public part of host key pair in ECDH

Response:

Field	Tag	Type	Description
pubB	0x1B	byte[65]	Public part of card key pair in ECDH
UID	0x0B	byte[4]	*Version 2.30 and later.* ISO 14443-3 unique identifier – for compatibility with devices where UID is hidden

4.7 Pending security delay

Version 1.16 and later.

In case a non-zero delay is set in Pause_Before_PIN2 personalization parameter, the host app should continuously resend the same command request until the card executes it. After each interim request,

the card will wait for ~1 second and respond with SW_NEED_PAUSE = 0x9789 and a TLV structure indicating remaining time of delay.

Version 1.21 and later.

The same mechanism applied in all commands to inform the host about all kinds of internal delays, including increasing delay on wrong PIN1 and PIN2.

Response:

Field	Tag	Type	Description
CID	0x01	byte[8]	Card ID (only in version 1.19 and earlier)
Pause	0x1C	Uint16	Remaining time, [10 x ms]
SavedInNVM	0x28	0 bytes	Version 2.30 and later. Optional flag indicating that COS has saved the value of remaining security delay in NVM.

Version 1.19 and earlier.

COS will return accordingly encrypted response if the command request has been encrypted with either fast or strong method.

Version 1.21 and later.

COS will return unencrypted response.

If communication is lost during the delay pending, then COS will immediately lose the state of countdown timer and restart it from the beginning after next command request.

Due to fluctuations of card's CPU clock and NFC field, COS can follow Pause_Before_PIN2 quite approximately in the range of -10% - +75%. Therefore, the host app should refrain from displaying the exact remaining time and use an indicative progress bar instead.

Version 2.30 and later.

For compatibility with iOS 13 and later versions, COS saves the value of remaining security delay in NVM every 5-10 seconds. COS additionally returns flag SavedInNVM if this value is stored. After the NFC session between card and terminal has been lost or reset, COS will continue the security delay countdown from the value stored in NVM if the first command received by COS coincide with the last one received before the session reset.

5. Personalization

These card parameters are defined by PERSONALIZE command:

CID, PIN1, PIN2, PIN3, Sign_Method,

NDEF, Card_Data,

Issuer_Data_PublicKey,

Issuer_Transaction_PublicKey,

Max_Signatures, Settings_Mask, Create_Wallet_At_Personalize,

Is_Activated, Pause_Before_PIN2

This section describes CID, NDEF, and Card_Data fields.

5.1 UID and CID

UID is 4-byte code required by ISO14443 standard. This code is randomly set only once during personalization process. It is never used in Tangem protocol except for NFC communication channel encryption key.

CID (Card ID) is a globally unique Tangem card number defined and printed (not embossed) by Tangem during personalization. It has 8 byte length and uses all hex digits (0..F). Example: 'AA00 0000 0001 6675'.

CID is encoded as SSSS NNNN NNNN NNNX, where:

- SSSS – batch number associated with the Issuer,
- NN..NN – unique card number within the batch,
- X – check digit according to Luhn algorithm, adapted to hex, see sample code in Appendix C.

5.2 NDEF records

During personalization, the issuer can define data of a static *NDEF* tag that will be emulated by COS if some NFC terminal (smartphone) will ever request such tag from the card. It is also possible to fully disable tag emulation. COS emulates NDEF format according to the NFC Forum standards:

- NFC Forum Type 4 Tag Operation Specification [NFCForum-TS-Type-4-Tag_2.0]
- NFC Data Exchange Format (NDEF) Technical Specification [NFCForum-TS-NDEF_1.0]

Default Tangem NDEF consists of two records:

1. URI, value - e.g. 'https://tangem.com'
   This record can be used to identify the card, even if there are no compatible applications installed on the smartphone. It is also used by iOS for background tag scanning.
2. AAR (Android application record), value – e.g. 'com.tangem.wallet' for native Tangem application (not used in iOS).

This record can be used to automatically open the corresponding App on the smartphone or to proceed to a correct issuer's application page on Google Play Store, if it's not installed on the smartphone.
This tag must be used with caution because an attacker distributing counterfeit Tangem cards could potentially exploit it to mislead the user in downloading a counterfeit application.

5.3 Card_Data

Structure and payload of this nested TLV field shall be defined together by Tangem and the issuer during personalization. App shall accordingly parse *Card_Data* field that is returned by READ_CARD command. The following minimum set of parameters is required to identify the card and the issuer, (native Tangem application will recognize these parameters):

Field	Tag	Type	Description
Batch_ID	0x81	byte[2], hex	Tangem internal manufacturing batch ID, should correspond with the data in Tangem's card list (see Card attestation and UID and CID sections)
Manufacture_Date_Time	0x82	byte[4]	Timestamp of manufacturing should correspond with the data in Tangem's card list (see Card attestation section). Format: Year (2 bytes) | Month (1 byte) | Day (1 byte)
Issuer_Name	0x83	string, utf8	Name of the issuer
Blockchain_Name	0x84	string, utf8	Name of the blockchain (once personalized, the target blockchain is fixed for the whole life cycle)

Manufacturer_Signature	0x86	byte[64]	Version 1.21 and later. Signature of CID with the MANUFACTURER_PRIVATE_KEY In some cases, signature of (CID || Card_PublicKey) with the MANUFACTURER_PRIVATE_KEY
ProductMask	0x8A	byte	Version 2.30 and later Mask of products enabled on card 0x01-Note, 0x02-Tag, 0x04-ID card

The issuer could append this list if more identifying parameters are required. For example, it would contain some data about a smart contact, if the issuer were going to store ERC20 tokens.

For example, native Tangem application will recognize the following additional parameters:

Field	Tag	Type	Description
Token_Symbol	0xA0	string, utf8	Name of the token
Token_Contract_Address	0xA1	string, utf8	Smart contract address
Token_Decimal	0xA2	string, utf8	Number of decimals in token value

6. Dynamic NDEF

Shall be supported ONLY on iOS versions 11 and 12. Other NFC smartphones shall not use this feature.

Tangem card supports a special wallet validation mechanism for smartphones that can only read NDEF tags, specifically iPhones 7, 8, 10 with iOS version 11 or later. Issuer can enable it by setting *Use_Dynamic_NDEF* flag in *Settings_Mask*.

In addition to two Tangem card NDEF records defined by personalization (see Personalization section for details), cards with dynamic NDEF will compose a third NDEF record for every new request of NDEF by smartphone. Dynamic NDEF has following structure:

- NDEF type: NFC Forum External
- Value: 'tangem.com:wallet'
- Payload:
    - Status_Word – command result (2 bytes, bigendian), same result codes as for all commands:
        i. 0x9000 – OK
        ii. 0x6A86 – TLV structure will not be returned because of some internal error, e.g. the card is protected by non-default *PIN1* (meaning that card and wallet data is simply not accessible).
    - TLV structure that contains:
        i. CID, Card_Data, Card_PublicKey, Health parameters, if there's no wallet has been created (card's Status is 'Empty'), or
        ii. A subset of parameters returned in responses of *READ_CARD* and *CHECK_WALLET* commands
        iii. VERSION 1.19 AND LATER included *Wallet_Signed_Hashes* field,
        iv. VERSION 1.21 AND LATER included *Settings_Mask* and *Max_Signatures* fields,

Dynamic NDEF works only if the card PIN1 is set to the default value. In this case, COS internally executes READ_CARD and CHECK_WALLET in order to compose the resulting TLV. The main point is that COS has to generate a random Challenge for *CHECK_WALLET* by itself, instead of receiving it from the application. This approach cannot guarantee full validation of the wallet, though provides a non-strict one that could be sufficient in some use cases. Some additional measures can alleviate the risk of attack of giving a counterfeit card or compromised wallet to a user who makes non-strict validation. First, the application can require to read dynamic NDEF of the same card many times to ensure the card generates different Challenges. In addition, there's always a risk for the attacked that the user has another fully functional NFC smartphone nearby.

Version 1.21 and later.

COS adapts to the limitation of tag response time in iOS 11.3.1. During the execution of CREATE_WALLET command, COS internally executes CHECK_WALLET twice to pre-compute and store two wallet signatures. If COS is unable to respond with the full TLV because of the host breaks the connection, it will respond with one of two precomputed signatures at the next attempt and toggle between two precomputed signature values at every next attempt. Although it is even less safe wallet validation method, it allows to protect against cheap clones using static programmable NFC tags.

Version 2.01 and later.

Issuer can disable precomputed NDEF by setting Disable_PrecomputedNDEF flag in Settings_Mask.

7. Activation

This mechanism allows safe physical transportation of the card through non-trusted environment. It disables most of card's commands from the moment of personalization and until the issuer confirms to COS that the card has been activated. To use activation mode, the issuer should disable *Is_Activated* flag during personalization. If it's done, the card will accept only these three commands:

- READ_CARD command will return only these fields:
  - *CID,*
  - *Manufacturer_ID,*
  - *Status,*
  - *Card_Data,*
  - *Settings_Mask,*
  - *Issuer_Data_PublicKey.*
- READ_ISSUER_DATA,
- ACTIVATE_CARD

ACTIVATE_CARD command provides guaranteed mutual update of activation state in both COS and issuer's activation back-end.

Activation process works as follows:

1. App calls READ_CARD command and recognizes Is_Activated flag is disabled,
2. App sends a request containing CID and Activation_Seed to issuer's activation back-end,
3. Issuer's activation back-end activates (and optionally top-ups) the card and sends a signed response to App,
4. App calls ACTIVATE_CARD command with the issuer's response as a parameter,
5. COS parses issuer's response and verifies its signatures; if it's OK, the COS renders itself to activated (fully functional) state.

In case of any interruption, App can safely restart this process from the beginning and repeat it until the card is activated. Therefore, it is guaranteed that the issuer's back-end will have completed all required activation operations before the card is switched to activated state.

8. Commands

Tangem COS provides 15 commands available to end user's host application and PERSONALIZE command that sets main parameters in the beginning of the life cycle.

The end user might optionally restrict access to the whole card and all commands by setting user's PIN1 code. By default, all cards have PIN1 set to '000000', so that every host application on any NFC smartphone could obtain card and wallet data. In the beginning of communication session, App should obtain card CID by calling READ_CARD command with hashed default PIN1. If the user has set a non-default PIN1, READ_CARD command will return Status Word SW_INVALID_PARAMS (= 0x6A86). In such case, the application should request correct PIN1 from the user and call READ_CARD command again,

this time with hashed PIN1 enter by the user. Once CID is obtained by READ_CARD command, the application can call all other commands that require CID and hashed PIN1 as parameters.

All cards will require submitting correct user's PIN2 code and (depending on personalization parameters) printed CVC code in order to sign a transaction or to perform other commands entailing change of the card state.App should ask the user to enter PIN2 and/or CVC before sending such commands to the card. There should be no default value for PIN2 code, if either CVC or very long Pause_Before_PIN2 delay are not used. See Security section for more details.

Commands can return the following Status Words in the result of execution:

- SW_PROCESS_COMPLETED = 0x9000
  - successful execution
- SW_INVALID_PARAMS = 0x6A86
  - wrong or not sufficient parameters in TLV request, or wrong PIN1/PIN2
- SW_ERROR_PROCESSING_COMMAND = 0x6286
  - internal error, incl. wrong issuer signature
- SW_INVALID_STATUS = 0x6985
  - command can not be executed in current card status
- SW_INS_NOT_SUPPORTED = 0x6D00
  - wrong command INS code

8.1 ACTIVATE_CARD

This command activates the card if it requires activation (Is_Activated flag is disabled). The application should go through an activation process in coordination with the issuer's activation back-end (see 'Activation' section for details).

Command INS code: **0xFE**

Parameters:

Field	Tag	Type	Description
CID	0x01	byte[8]	Unique Tangem card ID number
PIN1	0x10	byte[32]	Hashed user's PIN1 code to access the card, Default value should not be used with this command

Reset_PIN1	0x36	byte[1]	0 – Do not change PIN1 used for activation (for calling this command) 1 – Reset PIN1 to default value ('000000')
Activation_Signature	0x34	byte[64]	Concatenated (*CID* | *Activation_Seed* | *Reset_PIN1*) signed with *Issuer_Transaction_PrivateKey*; If the signature is successfully verified with *Issuer_Transaction_PublicKey*, then Is_Activated flag will be enabled

Response:

Same as in *READ_CARD* command

8.2 READ_CARD

This command returns all data about the card and the wallet, including unique card number (CID) that has to be submitted while calling all other commands. Therefore, READ_CARD should always be used in the beginning of communication session between host and Tangem card.

In order to obtain card's data, App should call READ_CARD command with correct PIN1 value as a parameter. The card will not respond if wrong PIN1 has been submitted. See Security section for details.

Command INS code: **0xF2**

Parameters:

Field	Tag	Type	Description

PIN1	0x10	byte[32]	Hashed user's PIN1 code to access the card. Default unhashed value: '000000'
Terminal_Public_Key	0x5C	byte[65]	VERSION 2.30 AND LATER Optional public key of linked host terminal (see 'Linked terminal' section) If the card was linked to the certain host terminal, Terminal_Public_Key would have to be passed. Otherwise, the card will be unlinked, and the next sign command will perform the security delay mechanism. See Linked terminal section for more details.

Response:

Field	Tag	Type	Description
CID	0x01	byte[8]	Unique Tangem card ID number
Manufacturer_Name	0x20	string, utf8	Name of Tangem card manufacturer *Fixed: 'Smart Cash'*
Status	0x02	byte	Current status of the card [1 - Empty, 2 - Loaded, 3- Purged]

Firmware_Version	0x80	string, utf8	Version of Tangem COS
Card_PublicKey	0x03	byte[65]	Public key that is used to authenticate the card against manufacturer's database. It is generated one time during card manufacturing. See Security section for more details.

Settings_Mask	0x0A	byte[2] **Version 2.01 and later.** byte[2] or byte[4]	Card settings defined by personalization (bit mask: 0 – Enabled, 1 – Disabled): **Is_Reusable = 0x0001** Defines what happens when user calls PURGE_WALLET command: *0 - Card will switch to Purged state* *1 - Card will switch to Empty state and let create a new wallet again* **Use_Activation = 0x0002** VERSION 2.01 AND LATER **Prohibit_Purge_Wallet = 0x0004** 0 – Card will accept PURGE_WALLET command 1 – Card will reject PURGE_WALLET command **Use_Block = 0x0008** **Allow_SET_PIN1 = 0x0010** Is user allowed to change PIN1 with SET_PIN command **Allow_SET_PIN2 = 0x0020** Is user allowed to change PIN2 with SET_PIN command **Use_CVC = 0x0040** All commands requiring PIN2 will also require additional CVC code printed on the card **Prohibit_Default_PIN1 = 0x0080** SET_PIN commands will not change PIN1 to default value (‘000000’). **Use_One_CommandAtTime = 0x0100** Card will execute only one command during one communication session, thus requiring user to physically take the card away from the host after each action (all commands except for READ_CARD). **Use_NDEF = 0x0200**

			Whether the card should emulate NDEF. In default configuration, two NDEF records are loaded during personalization: (1) Tangem web site address, (2) name of Android App package in Google Play Store. **Use_Dynamic_NDEF = 0x0400** 0 – Disable dynamic generation of NDEF for iOS. See Dynamic NDEF section for more details. 1 – Enable dynamic NDEF for iOS. **Smart_Security_Delay = 0x0800** Security delay Pause_Before_PIN2 will not be applied if PIN2 is not default. **Allow_Unencrypted = 0x1000** Whether the card supports unencrypted NFC communication. See NFC communication section for more details. **Allow_Fast_Encryption = 0x2000** Whether the card supports fast encrypted NFC communication. See NFC communication section for more details. **Protect_Issuer_Data_Against_Replay = 0x4000** VERSION 1.21 AND LATER 0 – No replay protection on write issuer data 1 – Enable replay protection on write issuer data (card will require additional Issuer_Data_Counter incremented on each write) **Allow_Select_Blockchain = 0x8000** VERSION 2.01 AND LATER 0 – Wallet elliptic curve and blockchain information stored during PERSONALIZE command and never change 1 – Wallet elliptic curve and blockchain information can be changed on CREATE_WALLET command

			Disable_Precomputed_NDEF = 0x00010000 0 – Enable precomputed dynamic NDEF to work around iPhone 7+ NFC bug. 1 – Disable precomputed dynamic NDEF. See Dynamic NDEF section for more details. **Skip_Security_Delay_If_Validated_By_Issuer = 0x00020000** VERSION 2.05 AND LATER 0 – Enforce security delay in SIGN command if the issuer validates the transaction (for signing methods 2, 3, 4 and 5). 1 – Skip security delay in SIGN command if the issuer validates the transaction (for signing methods 2, 3, 4 and 5). **Skip_Check_PIN2_and_CVC_If_Validated_By_Issuer = 0x00040000** VERSION 2.05 AND LATER 0 – Require and check PIN2 and CVC in SIGN command if the issuer validates the transaction (for signing method 2, 3, 4 and 5). 1 – Skip checking PIN2 and CVC in SIGN command if the issuer validates the transaction (for signing method 2, 3, 4 and 5). **Skip_Security_Delay_If_Validated_By_Linked_Terminal = 0x00080000** VERSION 2.30 AND LATER 1 - Store Terminal_PublicKey public key of linked terminal no each SIGN command, skip security delay if valid signature of transaction is made with Terminal_PrivateKey is provided in SIGN command **Restrict_Overwrite_Issuer_Extra_Data = 0x00100000** VERSION 2.30 AND LATER 1 – prohibit overwriting Issuer_Extra_Data

			Require_Term_Tx_Signature = 0x01000000 VERSION 2.30 AND LATER 0 – Skip checking terminal's signature when signing POS transaction 1 – Check terminal's signature when signing POS transaction **Require_Term_Cert_Signature = 0x02000000** VERSION 2.30 AND LATER 0 – Skip checking acquirer's signature of terminal certificate when signing POS transaction 1 – Check acquirer's signature of terminal certificate when signing POS transaction **Check_PIN3_on_Card = 0x04000000** VERSION 2.30 AND LATER 0 – Additionally encrypt POS transaction signature with key derived from PIN3 when the transaction amount exceeds PIN3 floor limit 1 – Require terminal to send PIN3 to card when the POS transaction amount exceeds *PIN3_Floor_Limit*
Card_Data	0x0C	byte[0..512]	Detailed information about card contents. Format is defined by the card issuer. Cards complaint with Tangem Wallet application should have TLV format described in Personalization section.
Issuer_Data_PublicKey	0x30	byte[65]	Public key that is used by the card issuer to sign Issuer_Data field. See Security section for more details.
Curve_ID	0x05	string, utf8	Explicit text name of the elliptic curve used for all wallet key operations. Supported curves: 'secp256k1', VERSION 2.01 AND LATER - 'ed25519'

Max_Signatures	0x08	uint32	Total number of signatures allowed for the wallet when the card was personalized.
Signing_Method	0x07	byte	Defines what data should be submitted to SIGN command. [0 - sign hash, 1 - sign raw transaction, 2 - sign hash signed by issuer, 3 - sign raw signed by issuer, 4 - sign hash signed by issuer and update Issuer_Data, 5 - sign raw signed by issuer and update Issuer_Data] **Version 2.05 and later.** Signing_Method can specify a set of allowed methods. In this case, the highest bit in Signing_Method value must be set to 1 and each of bits 0..5 must be set to enable corresponding signing methods. For example, if Signing_Method = 0x95, COS allows Signing_Method = 0, Signing_Method = 2, Signing_Method = 4; if Signing_Method = 0xBF – all methods are allowed. **Version 2.30 and later.** New Signing_Method value '6' – sign POS transactions (or bit #6 if multiple signing methods are allowed)
Pause_Before_PIN2	0x09	byte[2]	Delay in seconds before COS executes commands protected by PIN2.
Wallet_PublicKey	0x60	byte[65] **Version 2.01 and later** byte[65] or byte[32]	Public key of the blockchain wallet. Value returned only if the wallet has already been created by CREATE_WALLET command. See Security section for more details.

Wallet_Remaining_Signatures	0x62	uint32	Remaining number of SIGN operations before the wallet will stop signing transactions. Value returned only if the wallet has already been created by CREATE_WALLET command.
Wallet_Signed_Hashes	0x63	uint32	VERSION 1.16 AND LATER Total number of signed single hashes returned by the card in *SIGN* command responses since card personalization. Sums up array elements within all *SIGN* commands.
Health	0x0F	byte[1]	Any non-zero value indicates that the card experiences some hardware problems. User should withdraw the value to other blockchain wallet as soon as possible. Non-zero Health tag will also appear in responses of all other commands.
Is_Activated	0x3A	byte[1]	Whether the card requires issuer's confirmation of activation. 0 – card will require issuer's confirmation of activation, otherwise this field will not be returned (card is activated and operational). See *ACTIVATE_CARD* command for more details.
Activation_Seed	0x3B	byte[16]	A random challenge generated by PERSONALIZE command that should be signed and returned to COS by the issuer to confirm the card has been activated. See ACTIVATE_CARD command for more details. This field will not be returned if the card is activated.

Payment_Flow_Version	0x54	byte[2]	VERSION 2.30 AND LATER Version of POS payment scheme supported by COS ([0x02,0x01] for version 2.30) Returned only if SigningMethod '6' enabling POS transactions is supported by card.
User_Counter	0x2C	uint32	VERSION 2.30 AND LATER This value can be initialized by App and will be increased by COS with the execution of each SIGN command. For example, this field can store blockchain "nonce" for a quick one-touch transaction on POS terminals. Returned only if SigningMethod =6.
User_Protected_Counter	0x2D	uint32	VERSION 2.30 AND LATER This value can be initialized by App (with PIN2 confirmation) and will be increased by COS with the execution of each SIGN command. For example, this field can store blockchain "nonce" for a quick one-touch transaction on POS terminals. Returned only if SigningMethod =6.

Note: Parameters are included in the response depending on the card state:

- In the 'New' state: CID, Manufacturer_ID, Firmware_Version and Status fields are included,
- After personalization and before activation: additionally, Card_Data, Settings_Mask, Issuer_Data_PublicKey, Is_Activated, Activation_Seed fields,
- After personalization and activation, all other parameters are included, except for Wallet_PublicKey and Wallet_Remaining_Signatures.
- After a wallet is created, the Wallet_PublicKey and Wallet_Remaining_Signatures parameters are included.

8.3 CREATE_WALLET

This command will create a new wallet on the card having 'Empty' state. A key pair Wallet_PublicKey / *Wallet_PrivateKey* is generated and securely stored in the card. App will need to obtain *Wallet_PublicKey* from the response of CREATE_WALLET or READ_CARD commands and then transform it into an address of corresponding blockchain wallet according to a specific blockchain algorithm. *Wallet_PrivateKey* is never revealed by the card and will be used by SIGN and CHECK_WALLET. Remaining_Signature is set to Max_Signatures.

Command INS code: **0xF8**

Parameters:

Field	Tag	Type	Description
PIN1	0x10	byte[32]	Hashed user's PIN1 code to access the card. Default unhashed value: '000000'
CID	0x01	byte[8]	Unique Tangem card ID number
PIN2	0x11	byte[32]	Hashed user's PIN2 code for signing and state-changing operations. See Security section for more details.
CVC	0x19	byte[3]	Optional 3-digit code printed on the card. Required if Use_CVC flag is set in Settings_Mask.
VERSION 2.01 AND LATER If *Allow_Select_Blockchain* flag is set in *Settings_Mask* then the card allows changing type of blockchain in CREATE_WALLET command. The following parameters should must appended:			
Curve_ID	0x05	string, utf8	New explicit text name of the elliptic curve used for all wallet key operations. Supported curves: 'secp256k1', 'ed25519'
Card_Data	0x0C	byte[0..512]	Update to information about card contents in TLV format described in Personalization section. Tags that can be updated: Blockchain_Name, Token_Symbol, Token_Contract_Address, Token_Decimal If Card_Data contains other fields then command will fail and return error SW_INVALID_PARAMS.
Issuer_Data_Signature	0x33	byte[64]	Issuer's signature (with 'secp256k1' and *Issuer_Data_PrivateKey*) of SHA256-hashed Curve_ID concatenated with Card_Data: SHA256(Curve_ID | Card_Data).

Response:

Field	Tag	Type	Description
CID	0x01	byte[8]	Unique Tangem card ID number
Status	0x02	byte	Current status of the card [1 - Empty, 2 - Loaded, 3- Purged]
Wallet_PublicKey	0x60	byte[65] VERSION 2.01 AND LATER or byte[32]	Public key of a newly created blockchain wallet. See Security section for more details.

8.4 CHECK_WALLET

This command proves that the card possesses *Wallet_PrivateKey* corresponding to *Wallet_PublicKey*. Standard challenge/response scheme is used.

Command INS code: **0xF9**

Parameters:

Field	Tag	Type	Description
PIN1	0x10	byte[32]	Hashed user's PIN1 code to access the card. Default unhashed value: '000000'
CID	0x01	byte[8]	Unique Tangem card ID number
Challenge	0x16	byte[16]	Random challenge generated by host application
VERSION 2.01 AND LATER Self-attestation is supported. If the following additional parameter is specified, CHECK_WALLET will return Wallet_PublicKey signed by Card_PrivateKey:			
Public_Key_Challenge	0x14	byte[16] or byte[0]	Optional: Random challenge generated by host application. If specified, this field presents then card will return Card_Signature field in response

Response:

Field	Tag	Type	Description
CID	0x01	byte[8]	Unique Tangem card ID number
Salt	0x17	byte[16]	Random salt generated by the card
Wallet_Signature	0x61	byte[64]	Signature of hashed concatenated Challenge and Salt with *Wallet_PrivateKey* For 'secp256k1' curve SHA256 is used, signature of SHA256(Challenge | Salt) For 'ed25519' curve SHA512 is used, signature of SHA512(Challenge | Salt)
VERSION 2.01 AND LATER If Public_Key_Challenge had been submitted:			
Check_Wallet_Counter	0x64	byte[4]	Counter of CHECK_WALLET command executions. A very big value of this counter may indicate a hacking attempts.
Public_Key_Salt	0x15	byte[16]	Random salt generated by the card Optional, if in request presents non-empty Public_Key_Challenge
Card_Signature	0x04	byte[64]	Signature of hashed concatenated *Wallet_PublicKey*, *Public_Key_Challenge*, *Public_Key_Salt* with *Card_PrivateKey* Optional, if in request presents *Public_Key_Challenge* *SHA256(Wallet_PublicKey | Public_Key_Challenge | Public_Key_Salt)* signed with *Card_PrivateKey*, if in request presents non-empty Public_Key_Challenge or *SHA256(Wallet_PublicKey)* signed with *Card_PrivateKey*, if in request presents empty Public_Key_Challenge

8.5 SET_PIN

This command changes PIN1 and PIN2 passwords if it is allowed by Allow_SET_PIN1 and Allow_SET_PIN2 flags in Settings_Mask. Host application can submit unchanged passwords (New_PIN1 = PIN1 and New_PIN2 = PIN2) in order to check its correctness. Depending on the result, Status_Word in the command response will have these values:

SW_PINS_NOT_CHANGED = 0x9000

SW_PIN1_CHANGED = 0x9001

SW_PIN2_CHANGED = 0x9002

SW_PINS_CHANGED = 0x9003

VERSION 2.30 AND LATER

SW_PIN3_CHANGED = 0x9004

SW_PINS_CHANGED = 0x90XX, XX – changed PINs mask (0x01-PIN1, 0x02-PIN2, 0x04-PIN3)

Command INS code: **0xFA**

Parameters:

Field	Tag	Type	Description
PIN1	0x10	byte[32]	Hashed user's PIN1 code to access the card. Default unhashed value: '000000'
CID	0x01	byte[8]	Unique Tangem card ID number
PIN2	0x11	byte[32]	Hashed user's PIN2 code for signing and state-changing operations. See Security section for more details.
CVC	0x19	byte[3]	Optional 3-digit code printed on the card. Required if Use_CVC flag is set in Settings_Mask.
New_PIN1	0x12	byte[32]	New hashed user's PIN1 code to access the card.
New_PIN2	0x13	byte[32]	New hashed user's PIN2 code for signing and state-changing operations.

New_PIN3	0x1E	byte[32]	VERSION 2.30 AND LATER Optional new hashed user's PIN3 code for confirm POS operations.

Response:

Field	Tag	Type	Description
CID	0x01	byte[8]	Unique Tangem card ID number

8.6 SIGN

Depending on Signing_Method parameter defined during personalization, this command signs following data using *Wallet_PrivateKey*:

- array of transaction hashes (Signing_Method = 0)
- single raw transaction (Signing_Method = 1)

In case the card issuer supports 2FA (see in Security section), COS will firstly verify issuer's signature of transaction hash before signing transaction hash using *Wallet_PrivateKey*. Following data should be submitted to SIGN command in this case:

- array of transaction hashes signed by the issuer (Signing_Method = 2),
- single raw transaction signed by the issuer (Signing_Method = 3),

In case the card issuer requires the card to store additional data (e.g. wallet balance, SPV proof, timestamps, etc. – see details in Security section), COS will verify (1) issuer's signature of hash of concatenated transaction hashes and Issuer_Data hash, and (2) issuer's signature of Issuer_Data hash, and then proceed to signing transaction hash using *Wallet_PrivateKey*. Following data should be submitted to SIGN command in this case:

- array of transaction hashes and issuer data signed by the issuer (Signing_Method = 4),
- single raw transaction and issuer data signed by the issuer (Signing_Method = 5),

The raw transaction will be hashed by a hash function 'Hash_Name' before signing. In the generic COS version described here, raw transactions are not parsed, analyzed or transformed by COS. Simultaneous signing of array of hashes in a single SIGN command is required to support Bitcoin-type multi-input blockchains. SIGN command will return a corresponding array of signatures. If COS cannot validate issuer's signature where it is required, SIGN command will return Status word SW_ERROR_PROCESSING_COMMAND= 0x6286.

After data is signed and SIGN response is successfully sent to the host, COS decreases Wallet_Remaining_Signatures counter by 1. When Remaining_Signatures counter reaches zero, SIGN command will no longer generate any signatures for the wallet. COS always stores cached signature(s) generated by the last SIGN command and return this cached result if SIGN command is called again

with empty parameters. In this way, the signature(s) can be restored in case App has lost the previous SIGN response for any reason.

Version 2.05 and later.

Signing_Method can specify a set of allowed methods. In this case, the highest bit in Signing_Method value must be set to 1 and each of bits 0..5 must be set to enable corresponding signing methods: bit 0 (lowest) – allow Signing_Method = 0, bit 1 – allow Signing_Method = 1 and so on. For example, if Signing_Method = 0x95 COS allow Signing_Method = 0, Signing_Method = 2, Signing_Method = 4; if Signing_Method = 0xBF – all methods are allowed. If more than one signing method is allowed, the application can select a method by providing an appropriate set of parameters for SIGN command.

Version 2.30 and later.

New Signing_Method = 6 – "one-touch" signing of transaction received from trusted POS terminal (see 'Acquirer' section and 'POS mode' below)

[SIGN CONTINUED]

8.6.1 Wallet Mode

(Signing_Methods = 0, 1, 2, 3, 4, 5)

Command INS code: **0xFB**

Parameters:

Field	Tag	Type	Description
CID	0x01	byte[8]	Unique Tangem card ID number
PIN1	0x10	byte[32]	Hashed user's PIN1 code to access the card. Default unhashed value: '000000'
PIN2	0x11	byte[32]	Hashed user's PIN2 code for signing and state-changing operations. See Security section for more details.
CVC	0x19	byte[3]	Optional 3-digit code printed on the card. Required if Use_CVC flag is set in Settings_Mask.
Transaction_ Hash_Size	0x51	byte[1]	Length of a single hash to be signed (0x20 for SHA256). Required if Signing_Method = 0, 2, 4

Transaction_ Hash	0x50	byte[n*hash _size]	Concatenated array of hashes to be signed. Required if Signing_Method = 0, 2, 4
Transaction_ Raw	0x52	byte[1..512]	Raw transaction to be hashed and signed. Required if Signing_Method = 1, 3, 5
Hash_Name	0x06	byte[5..9]	Text name of hash function to hash raw transaction, supported value are ‘sha-256’, ‘sha-1’, ‘sha-224’, ‘sha-384’, ‘sha-512’. It’s possible to specify a double hashing by add ‘x2’ to the hash function name. Required if Signing_Method = 1, 3, 5
Issuer_Data	0x32	byte[1..512]	Some issuer’s data that must be stored in the card at the moment of signing the transaction. May contain wallet balance, SPV proof, timestamps, etc. This data is never parsed, analyzed or transformed by COS. See Security section for more details.
Issuer_Data_ Counter	0x35	int[4]	VERSION 1.21 AND LATER An internal counter received from READ_ISSUER_DATA command. Must be signed by the issuer together with Issuer_Data. Required if Signing_Method = 4, 5.

Issuer_Transaction_Signature	0x34	byte[64]	*If Signing_Method = 2* issuer's signature of a hash of concatenated array of transaction hashes: SHA256(tx_hash_1 | … | tx_hash_n) *If Signing_Method = 3* issuer's signature of a hash of raw transaction: SHA256(tx_raw) *If Signing_Method = 4* issuer's signature of a hash of concatenated array of transaction hashes and Issuer_Data: SHA256(tx_hash_1 | … | tx_hash_n | (Issuer_Data)) *If Signing_Method = 5* issuer's signature of a hash of concatenated hash of raw transaction and Issuer_Data: SHA256(SHA256(tx_raw) | (Issuer_Data))
Issuer_Data_Signature	0x33	byte[64]	Required if Signing_Method = 4, 5. **Version 1.19 and earlier.** Issuer's signature of SHA256-hashed Issuer_Data concatenated with CID: SHA256(CID | Issuer_Data) **Version 1.21 and later.** Issuer's signature of SHA256-hashed Issuer_Data concatenated with CID and Issuer_Data_Counter : SHA256(CID | Issuer_Data | Issuer_Data_Counter)
Terminal_PublicKey	0x5C	byte[65]	VERSION 2.30 AND LATER Optional public key of linked host terminal (see 'Linked terminal' section) Only for Signing_Method = 0, 1

Terminal_Transaction_Signature	0x57	byte[64]	VERSION 2.30 AND LATER Optional transaction signature by linked host terminal (see 'Linked terminal' section) Only for Signing_Method = 0, 1 *If Signing_Method = 0* signature of a hash of concatenated array of transaction hashes: SHA256(tx_hash_1 | … | tx_hash_n) *If Signing_Method = 1* signature of a hash of raw transaction: SHA256(tx_raw)

Response:

Field	Tag	Type	Description
CID	0x01	byte[8]	Unique Tangem card ID number
Signature	0x40	byte[n*64]	Array of resulting signatures that App should embed into a raw transaction according to a transaction format of an appropriate blockchain.
Wallet_Remaining_Signatures	0x62	uint32	Remaining number of SIGN operations before the wallet will stop signing transactions.
Wallet_Signed_Hashes	0x63	uint32	VERSION 1.16 AND LATER Total number of signed single hashes returned by the card in SIGN command responses since card personalization. Sums up array elements within all SIGN commands.

8.6.2 POS mode

VERSION 2.30 AND LATER

(Signing_Method = 6)

Command INS code: **0xFB**

Parameters:

Field	Tag	Type	Description
CID	0x01	byte[8]	Unique Tangem card ID number
PIN1	0x10	byte[32]	Hashed user's PIN1 code to access the card. Default unhashed value: '000000'
Payment_Flow_Version	0x54	byte[2]	Version of POS payment scheme the same, as supported by COS ([0x02,0x01] for version 2.30)
Transaction_Hash_Size	0x51	byte[1]	Length of a single hash to be signed (0x20 for SHA256)
Terminal_Certificate	0x55	byte[...]	Terminal certificate – nested TLV data contained fields: *Terminal_Id* 0x5A byte[0..32] terminal ID string *Terminal_Param* 0x5B uint32 mask of terminal params (RFU) *Terminal_PublicKey* 0x5C byte[65] public key of terminal *Terminal_Signature* 0x5D byte[64] acquirer signature of terminal certificate. ECDSA-signature of SHA-256(*Terminal_Certificate*) with *Terminal_PrivateKey*
Transaction_Amount	0x53	BigInteger	Amount of transaction in the same units as *PIN3_Floor_Limit* specified during personalization, big-endian format
Transaction_Extra_Hash	0x56	byte[..]	Additional hash of extra data to be signed

Terminal_Transaction_Signature	0x57	byte[64]	Optional transaction signature by terminal signature of a hash of concatenated array of transaction hashes: SHA256(Transaction_Hash | Transaction_Amount | Transaction_Extra_Hash)
PIN3	0x1D	byte[32]	Optional PIN3 (mandatory if Check_PIN3_on_Card flag set in Settings_Mask)

Response:

Field	Tag	Type	Description
CID	0x01	byte[8]	Unique Tangem card ID number
CrEx_Salt	0x65	byte[32]	Random salt (see “One-touch” POS payments)
CrEx_CryptedMessage	0x67	uint32	Crypted message (see “One-touch” POS payments)
CrEx_HMAC	0x68	byte[32]	Message verification code (see “One-touch” POS payments)
CrEx_RequirePIN3	0x69		Optional flag signalize that terminal need request user enter PIN3 to confirm transaction

8.7 READ_ISSUER_DATA

This command returns 512-byte Issuer_Data field and its issuer’s signature. See details in Issuers section.

Issuer_Data is never changed or parsed by the executable code the Tangem COS. The issuer defines purpose of use, format and payload of Issuer_Data. For example, this field may contain information about wallet balance signed by the issuer or additional issuer’s attestation data.

Command INS code: ***0xF7***

Parameters:

Field	Tag	Type	Description
CID	0x01	byte[8]	Unique Tangem card ID number
PIN1	0x10	byte[32]	Hashed user's PIN1 code to access the card. Default unhashed value: '000000'

Response:

Field	Tag	Type	Description
CID	0x01	byte[8]	Unique Tangem card ID number
Issuer_Data	0x32	byte[1..512]	Data defined by issuer
Issuer_Data_Signature	0x33	byte[64]	Issuer's signature of Issuer_Data with *Issuer_Data_PrivateKey* **Version 1.19 and earlier** Issuer's signature of SHA256-hashed CID concatenated with Issuer_Data: SHA256(CID | Issuer_Data) **Version 1.21 and later** When flag Protect_Issuer_Data_Against_Replay set in Settings_Mask then signature of SHA256-hashed CID Issuer_Data concatenated with and Issuer_Data_Counter : SHA256(CID | Issuer_Data | Issuer_Data_Counter)
Issuer_Data_Counter	0x35	int[4]	VERSION 1.21 AND LATER An optional counter that protect issuer data against replay attack. When flag Protect_Issuer_Data_Against_Replay set in Settings_Mask then this value is mandatory and must increase on each execution of WRITE_ISSUER_DATA command.

8.7.1 Read issuer extra data

VERSION 2.30 AND LATER

This command retrieves Issuer_Extra_Data field and its issuer's signature. See details in 'Issuers' section.

Issuer_Extra_Data is never changed or parsed by the executable code the Tangem COS. The issuer defines purpose of use, format and payload of Issuer_Extra_Data. For example, this field may contain photo or biometric information for ID card product. Because of the large size of Issuer_Extra_Data, a series of these commands have to be executed to read the entire Issuer_Extra_Data.

Command INS code: **0xF7**

Parameters:

Field	Tag	Type	Description
CID	0x01	byte[8]	Unique Tangem card ID number
PIN1	0x10	byte[32]	Hashed user's PIN1 code to access the card. Default unhashed value: '000000'
Mode	0x23	byte	0 or omitted – return Issuer_Data 1 – return Issuer_Extra_Data
Offset	0x24	uint16	Offset in Issuer_Extra_Data to requested data part. Only for reading Issuer_Extra_Data (Mode=1)

Response:

Field	Tag	Type	Description
CID	0x01	byte[8]	Unique Tangem card ID number
Size	0x25	uint16	Size of all Issuer_Extra_Data field Returns only for reading Issuer_Extra_Data (Mode=1) and Offset=0

Issuer_Data or Issuer_Extra_Data	0x32	byte[1..]	Data defined by issuer (all Issuer_Data field for Mode=0 or part of Issuer_Extra_Data for Mode=1)
Issuer_Data_Signature or Issuer_Extra_Data_Signature	0x33	byte[64]	Issuer's signature of Issuer_Data (Mode=0) or Issuer_Extra_Data (Mode=1) with *Issuer_Data_PrivateKey* If flags Protect_Issuer_Data_Against_Replay and Restrict_Overwrite_Issuer_Extra_Data aren't set in Settings_Mask then signature of SHA256-hashed CID concatenated with Issuer_Data or Issuer_Extra_Data: SHA256(CID | Issuer_Data) or SHA256(CID | Issuer_Extra_Data). Otherwise the signature of SHA256-hashed CID concatenated with Issuer_Data or Issuer_Extra_Data and Issuer_Data_Counter or Issuer_Extra_Data_Counter: SHA256(CID | Issuer_Data | Issuer_Data_Counter) or SHA256(CID | Issuer_Extra_Data | Issuer_Data_Counter) For Mode=1 this value return only with last part of Issuer_Extra_Data and size of data part is determined by free space in communication buffer.
Issuer_Data_Counter or Issuer_Extra_Data_Counter	0x35	int[4]	An optional counter that protect issuer data against replay attack. When flags Protect_Issuer_Data_Against_Replay or Restrict_Overwrite_Issuer_Extra_Data set in Settings_Mask then this value is mandatory and must increase on each execution of WRITE_ISSUER_DATA command. There are two independent counters in COS to protect both Issuer_Data and Issuer_Extra_Data. For Mode=1 this value return only with last part of Issuer_Extra_Data.

8.8 WRITE_ISSUER_DATA

This command write 512-byte Issuer_Data field and its issuer's signature. See details in Issuers section.

Issuer_Data is never changed or parsed by the executable code the Tangem COS. The issuer defines purpose of use, format and payload of Issuer_Data. For example, this field may contain information about wallet balance signed by the issuer or additional issuer's attestation data.

Command INS code: **0xF6**

Parameters:

Field	Tag	Type	Description
CID	0x01	byte[8]	Unique Tangem card ID number
PIN1	0x10	byte[32]	Hashed user's PIN1 code to access the card. Default unhashed value: '000000'
Issuer_Data	0x32	byte[1..512]	Data defined by issuer
Issuer_Data_Signature	0x33	byte[64]	Issuer's signature of Issuer_Data with *Issuer_Data_PrivateKey* **Version 1.19 and earlier** Issuer's signature of SHA256-hashed Issuer_Data concatenated with CID: SHA256(CID | Issuer_Data) **Version 1.21 and later** When flag Protect_Issuer_Data_Against_Replay set in Settings_Mask then issuer's signature of SHA256-hashed Issuer_Data concatenated with CID and Issuer_Data_Counter : SHA256(CID | Issuer_Data | Issuer_Data_Counter)

Issuer_Data_Counter VERSION 1.21 AND LATER	0x35	int[4]	An optional counter that protect issuer data against replay attack. When flag Protect_Issuer_Data_Against_Replay set in Settings_Mask then this value is mandatory and must increase on each execution of WRITE_ISSUER_DATA command.

Response:

Field	Tag	Type	Description
CID	0x01	byte[8]	Unique Tangem card ID number

8.8.1 Write issuer extra data

VERSION 2.30 AND LATER

This command writes Issuer_Extra_Data field and its issuer's signature. See details in Issuers section.

Issuer_Extra_Data is never changed or parsed by the executable code the Tangem COS. The issuer defines purpose of use, format and payload of Issuer_Extra_Data. For example, this field may contain a photo or biometric information for ID card products. Because of the large size of Issuer_Extra_Data, a series of these commands have to be executed to write entire Issuer_Extra_Data.

Command INS code: **0xF6**

Parameters:

Field	Tag	Type	Description
CID	0x01	byte[8]	Unique Tangem card ID number
PIN1	0x10	byte[32]	Hashed user's PIN1 code to access the card. Default unhashed value: '000000'
Mode	0x23	byte	0 or omitted – write Issuer_Data 1 – start write Issuer_Extra_Data 2 – write part of Issuer_Extra_Data 3 – finalize write Issuer_Extra_Data

Size	0x25	uint16	Size of all Issuer_Extra_Data field Only for start writing Issuer_Extra_Data (Mode=1)
Issuer_Data	0x32	byte[1..]	Data defined by issuer (all Issuer_Data field for Mode=0 or part of Issuer_Extra_Data for Mode=2)
Issuer_Data_Signature	0x33	byte[64]	Issuer's signature with *Issuer_Data_PrivateKey* of: *for Mode=0:* When flag Protect_Issuer_Data_Against_Replay set in Settings_Mask then issuer's signature of SHA256-hashed Issuer_Data concatenated with CID and Issuer_Data_Counter : SHA256(CID | Issuer_Data | Issuer_Data_Counter) *for Mode=1:* If flags Protect_Issuer_Data_Against_Replay and Restrict_Overwrite_Issuer_Extra_Data are not set in Settings_Mask then signature of SHA256-hashed CID concatenated with Size: SHA256(CID | Size). Otherwise signature of SHA256-hashed CID concatenated with Issuer_Extra_Data_Counter and Size: SHA256(CID | Issuer_Data_Counter | Size) *for Mode=3:* If flags Protect_Issuer_Data_Against_Replay and Restrict_Overwrite_Issuer_Extra_Data aren't set in Settings_Mask then signature of SHA256-hashed CID concatenated with Issuer_Extra_Data: SHA256(CID | Issuer_Extra_Data). Otherwise the signature of SHA256-hashed CID concatenated with Issuer_Extra_Data and Issuer_Extra_Data_Counter: SHA256(CID | Issuer_Extra_Data | Issuer_Data_Counter)

Issuer_Data_Counter	0x35	int[4]	An optional counter that protect issuer data against replay attack. When flag Protect_Issuer_Data_Against_Replay or Restrict_Overwrite_Issuer_Extra_Data set in Settings_Mask then this value is mandatory and must increase on each execution of WRITE_ISSUER_DATA command.
Offset	0x24	uint16	Offset in Issuer_Extra_Data to requested data part. Only for writing part of Issuer_Extra_Data (Mode=2)

Response:

Field	Tag	Type	Description
CID	0x01	byte[8]	Unique Tangem card ID number

8.9 VERIFY_CODE

This command challenges the card to prove integrity of COS binary code. For this purpose, App should have a special 'hash library' publicly provided by Tangem. Approx. size of the library is 5Mb. It contains ~150.000 precalculated SHA512 hashes of COS binary code segments.

VERIFY_CODE command internally reads a segment of COS binary code beginning at Code_Page_Address and having length of [64 x Code_Page_Count] bytes. Then it appends Challenge to the code segment, calculates resulting hash and returns it in the response. The application needs to ensure that returned hash coincides with the one stored in the hash library.

Command INS code: **0xF5**

Parameters:

Field	Tag	Type	Description
CID	0x01	byte[8]	Unique Tangem card ID number
PIN1	0x10	byte[32]	Hashed user's PIN1 code to access the card. Default unhashed value: '000000'
Hash_Name	0x06	byte[5..9]	Text name of hash function, supported value are 'sha-256', 'sha-1', 'sha-224', 'sha-384', 'sha-512', 'crc-16'

Code_Page_Address	0x40	byte[4]	Value from 0 to ~3000.
Code_Page_Count	0x41	byte[2]	Number of 32-byte pages to read: from 1 to 5, or 0 (only for Code_Page_Address=0) When Code_Page_Address=0 and Code_Page_Count=0 then all Binary Code used to calculate hash
Challenge	0x16	byte[16]	Additional challenge value from 1 to 10

Response:

Field	Tag	Type	Description
CID	0x01	byte[8]	Unique Tangem card ID number
Code_Hash	0x42	byte[32]	Resulting hash SHA256(Firmware_Version | Challenge | Code_Page_Address | Code_Page_Count | Binary Code)

8.10 VERIFY_CARD

This command is used for card attestation. See details in Security section.

The application shall call VERIFY_CARD command to ensure the card has not been counterfeited. By using standard challenge-response scheme, the card proves possession of Card_PrivateKey that corresponds to Card_PublicKey returned by READ_CARD command.

Command INS code: **0xF3**

Parameters:

Field	Tag	Type	Description
CID	0x01	byte[8]	Unique Tangem card ID number
PIN1	0x10	byte[32]	Hashed user's PIN1 code to access the card. Default unhashed value: '000000'
Challenge	0x16	byte[16]	Random challenge generated by host application

Response:

Field	Tag	Type	Description
CID	0x01	byte[8]	Unique Tangem card ID number
Salt	0x17	byte[16]	Random salt generated by the card
Card_Signature	0x04	byte[64]	Hashed concatenated Challenge and Salt: SHA256(Challenge | Salt) signed with Card_PrivateKey

8.11 VALIDATE_CARD

This is an optional command that the issuer can support if there is a real risk of mass counterfeiting by making multiple clones of a single card. This can be the case for transferrable Tangem cards that are almost never redeemed by users.

The issuer has to have a back-end service storing and updating a counter value (Card_Validation_Counter) for each card (CID). This function should also be supported by the issuer's application.

The application may occasionally call VALIDATE_CARD command to ensure that there's only one card having this CID is circulating out there. VALIDATE_CARD will increase COS internal Card_Validation_Counter by 1 and sign the new value with Card_PrivateKey. Then the application should submit increased Card_Validation_Counter and its signature to issuer's card validation back-end (server). The server should verify the signature and update Card_Validation_Counter value if previous value is less than the new one. If the server reveals that submitted Card_Validation_Counter value is less than previous value, then the card having this CID is deemed compromised and should not be accepted by the application.

Command INS code: ***0xF4***

Parameters:

Field	Tag	Type	Description
CID	0x01	byte[8]	Unique Tangem card ID number
PIN1	0x10	byte[32]	Hashed user's PIN1 code to access the card. Default unhashed value: '000000'
PIN2	0x11	byte[32]	Hashed user's PIN2 code for signing and state-changing operations. See Security section for more details.

Response:

Field	Tag	Type	Description
CID	0x01	byte[8]	Unique Tangem card ID number
Card_Validation_Counter	0x18	byte[2]	Internal validation counter
Card_Signature	0x04	byte[64]	Hashed Card_Validation_Counter signed with Card_PrivateKey

8.12 PURGE_WALLET

This command deletes all wallet data. If *Is_Reusable* flag is enabled during personalization, the card changes state to 'Empty' and a new wallet can be created by CREATE_WALLET command. If *Is_Reusable* flag is disabled, the card switches to 'Purged' state. 'Purged' state is final, it makes the card useless.

Version 2.01 and later.

This command can be prohibited during the personalization stage by set flag Prohibit_Purge_Wallet in Settings_Mask.

Command INS code: **0xFC**

Parameters:

Field	Tag	Type	Description
CID	0x01	byte[8]	Unique Tangem card ID number
PIN1	0x10	byte[32]	Hashed user's PIN1 code to access the card. Default unhashed value: '000000'
PIN2	0x11	byte[32]	Hashed user's PIN2 code for signing and state-changing operations. See Security section for more details.

Response:

Field	Tag	Type	Description
CID	0x01	byte[8]	Unique Tangem card ID number

Status	0x02	byte	Current status of the card [1 - Empty, 2 - Loaded, 3- Purged]

8.13 READ_USER_DATA

VERSION 2.30 AND LATER

This command returns two up to 512-byte User_Data, User_Protected_Data and two counters User_Counter and User_Protected_Counter fields.

User_Data and User_ProtectedData are never changed or parsed by the executable code the Tangem COS. The App defines purpose of use, format and it's payload. For example, this field may contain cashed information from blockchain to accelerate preparing new transaction.

User_Counter and User_ProtectedCounter are counters, that initial values can be set by App and increased on every signing of new transaction (on SIGN command that calculate new signatures). The App defines purpose of use. For example, this fields may contain blockchain nonce value.

Command INS code: ***0xE1***

Parameters:

Field	Tag	Type	Description
CID	0x01	byte[8]	Unique Tangem card ID number
PIN1	0x10	byte[32]	Hashed user's PIN1 code to access the card. Default unhashed value: '000000'

Response:

Field	Tag	Type	Description
CID	0x01	byte[8]	Unique Tangem card ID number
User_Data	0x2A	byte[1..512]	Data defined by user's App
User_ProtectedData	0x2B	byte[1..512]	Data defined by user's App (confirmed by PIN2)
User_Counter	0x2C	int[4]	Counter initialized by user's App and increased on every signing of new transaction

User_ProtectedCounter	0x2D	int[4]	Counter initialized by user's App (confirmed by PIN2) and increased on every signing of new transaction

8.14 WRITE_USER_DATA

VERSION 2.30 AND LATER

This command write some of User_Data, User_ProtectedData, User_Counter and User_ProtectedCounter fields.

User_Data and User_ProtectedData are never changed or parsed by the executable code the Tangem COS. The App defines purpose of use, format and it's payload. For example, this field may contain cashed information from blockchain to accelerate preparing new transaction.

User_Counter and User_ProtectedCounter are counters, that initial values can be set by App and increased on every signing of new transaction (on SIGN command that calculate new signatures). The App defines purpose of use. For example, this fields may contain blockchain nonce value.

Writing of User_Counter and User_Data protected only by PIN1. User_ProtectedCounter and User_ProtectedData additionaly need PIN2 to confirmation.

Command INS code: **0xE0**

Parameters:

Field	Tag	Type	Description
CID	0x01	byte[8]	Unique Tangem card ID number
PIN1	0x10	byte[32]	Hashed user's PIN1 code to access the card. Default unhashed value: '000000'
PIN2	0x11	byte[32]	Optional hashed user's PIN2 code to confirm write User_ProtectedData and User_ProtectedCounter
User_Data	0x2A	byte[1..512]	Optional data defined by user's App
User_ProtectedData	0x2B	byte[1..512]	Optional data defined by user's App (PIN2 is mandatory)
User_Counter	0x2C	int[4]	Optional new user counter value
User_Counter	0x2D	int[4]	Optional new protected user counter value

Response:

Field	Tag	Type	Description
CID	0x01	byte[8]	Unique Tangem card ID number

9 Appendix A – Dynamic NDEF

9.1 General description

Card supports reading data in NDEF format in accordance with the standards of the NFC Forum:

- NFC Forum Type 4 Tag Operation Specification [NFCForum-TS-Type-4-Tag_2.0]
- NFC Data Exchange Format (NDEF) Technical Specification [NFCForum-TS-NDEF_1.0]

NDEF contains three records:

1. URI (value 'tangem.com')
2. AAR (android application record, value 'com.tangem.wallet') – not used in iOS
3. Dynamically generated record:

- Type – NFC Forum External type (value 'tangem.com:wallet')
- Content:
- Status – two bytes (0x9000 – OK, 0x6A86 – card has PIN code set up, so TLV structure will not be given)
- TLV data structure:
    - CardID – unique car number
        - Tag: 0x01
        - Length: 8
    - Firmware - card firmware version
        - Tag: 0x80
        - Length: 4 – 10
        - UTF-8 string
    - SettingsMask – card settings (see full manual)
        - Tag: 0x0A
        - Length: 2
    - Card_Data – nested TLV structure with card properties
        - Tag: 0x0C
        - Length: 0 – 512
        - Format:
            - Batch_ID – batch number
                - Tag: 0x81
                - Length: 2
            - Manufacture_Date_Time - manufacture date and time

        - Tag: 0x82
        - Length: 4
    - Issuer_Name - card issuer name
        - Tag: 0x83
        - UTF-8 string
    - Blockchain_Name – the name of blockchain, which key card holds
        - Tag: 0x84
        - UTF-8 string
    - Token_Symbol – (optional) ERC20 token sybol (e.g., 'SEED')
        - Tag: 0xA0
        - UTF-8 string
    - Token_Contract_Address – (optional) smart contract adress for ERC20 token
        - Tag: 0xA1
        - UTF-8 string
    - Token_Decimal – (optional) number of decimal places for ERC20 token (usually 18)
        - Tag: 0xA2
        - 1 Byte
    - Manufacturer_Signature – CID or (CID || Card_PublicKey) signed with manufacturer's secret key
        - Tag: 0x86
        - Length: 64
        - Format [R[32], S[32]] for elliptical curve signature
- Card_PublicKey – card public key
    - Tag: 0x03
    - Length: 65
- Wallet_PublicKey - public key of the blockchain wallet
    - Tag: 0x60
    - Length: 65
    - Format [0x04, X[32], Y[32]], elliptical curve secp256k1
- MaxSignatures – initial number of allowed transaction signatures (set on personalization)
    - Tag: 0x08
    - Length: 4
- RemainingSignatures – remaining number of allowed transaction signatures
    - Tag: 0x62
    - Length: 4
- SignedHashes – number of hashes signed after personalization (there can be severeal hases in one transaction)
    - Tag: 0x63
    - Length: 4

- Challenge – first part of a message signed by card
  - Tag: 0x16
  - Length: 16
- Salt – second part of a message signed by card
  - Tag: 0x17
  - Length: 16
- Wallet_Signature – [Challenge, Salt] SHA256 signature signed with Wallet_PrivateKey
  - Tag: 0x61
  - Length: 64
- Format [R[32],S[32]] for elliptical curve signature
- Health – card state (0 – OK, other value – card is damaged)
  - Tag: 0x0F
  - Length: 1

Note: If the card has no wallet (Status is set to 'Empty'), then the record will contain only the following values: CardID, Firmware, Card_Data, Card_PublicKey, Health.

9.2 Example (card with PIN set):

- Record #1: URI record
  - Type Name Format: NFC Forum well-known type(1),
  - Short Record,
  - Type: "U"
  - Payload length: 11 bytes
  - Payload data: 01 74 61 6E 67 65 6D 2E 63 6F 6D ('tangem.com')
- Record #2: Android Application record
  - Type Name Format: NFC Forum external type (4),
  - Short Record,
  - Type: "android.com:pkg"
  - Payload length: 17 bytes
  - Payload data: 63 6F 6D 2E 74 61 6E 67 65 6D 2E 77 61 6C 6C 65 74 ('com.tangem.wallet')
- Record #3: NFC Forum external type record
  - Type Name Format: NFC Forum external type (4)
  - Short Record
  - Type: "tangem.com:wallet"
  - Payload length: 2 bytes
  - Payload data: 6A 86

9.3 Example (before wallet creation):

- Record #1: URI record
  - Type Name Format: NFC Forum well-known type(1),
  - Short Record,

- type: "U"
- Payload length: 11 bytes
- Payload data: 01 74 61 6E 67 65 6D 2E 63 6F 6D (‘tangem.com’)

- Record #2: Android Application record
  - Type Name Format: NFC Forum external type (4),
  - Short Record,
  - type: "android.com:pkg"
  - Payload length: 17 bytes
  - Payload data: 63 6F 6D 2E 74 61 6E 67 65 6D 2E 77 61 6C 6C 65 74 (‘com.tangem.wallet’)
- Record #3: NFC Forum external type record
  - Type Name Format: NFC Forum external type (4)
  - Short Record
  - type: "tangem.com:wallet"
  - Payload length: 120 bytes
  - Payload data:

```
90 00 01 08 CB 01 00 00 00 00 00 04 80 06 31 2E
32 38 72 00 0C 93 81 02 00 15 82 04 07 E2 07 1B
83 0B 53 55 50 45 52 42 4C 4F 4F 4D 00 84 03 45
54 48 A0 04 53 45 45 44 A1 2A 30 78 34 45 37 42
64 38 38 45 33 39 39 36 66 34 38 45 32 61 32 34
44 31 35 45 33 37 63 41 34 43 30 32 42 34 44 31
33 34 64 32 A2 01 12 86 40 31 AE DE CF 5E AE 50
1D 2E EB 07 E5 5C 6F 1B DE FC D3 DC BE EA 9A EA
C7 95 96 4B 60 EA DA BB 11 AB 92 62 F2 D0 14 E8
29 E5 DD 3C 32 20 45 3C 1C 68 D0 D1 19 27 77 1B
1F 72 7C 21 52 86 C8 50 31 03 41 04 1E 2C 7E 19
3A C0 D9 25 ED AC 5E 7B 97 6E 82 0E ED D7 23 E4
02 C4 14 43 19 17 16 2D 44 25 FF F5 93 07 14 12
3F AB D5 83 D5 C0 9E 5A C0 40 5F 16 17 9F 47 4D
41 8C B2 32 F6 1C 9E 6B 57 3B 32 E4 0F 01 00
```

9.4 Example (after wallet creation):

- Record #1: URI record
  - Type Name Format: NFC Forum well-known type(1),
  - Short Record,
  - type: "U"
  - Payload length: 11 bytes
  - Payload data: 01 74 61 6E 67 65 6D 2E 63 6F 6D (‘tangem.com’)
- Record #2: Android Application record
  - Type Name Format: NFC Forum external type (4),
  - Short Record,
  - type: "android.com:pkg"
  - Payload length: 17 bytes

- Payload data: 63 6F 6D 2E 74 61 6E 67 65 6D 2E 77 61 6C 6C 65 74 ('com.tangem.wallet')

- Record #3: NFC Forum external type record
  - Type Name Format: NFC Forum external type (4)
  - Type: "tangem.com:wallet"
  - Payload length: 295 bytes
  - Payload data:

```
90 00 01 08 CB 01 00 00 00 00 00 04 80 06 31 2E
32 38 72 00 0A 02 7E 31 0C 93 81 02 00 15 82 04
07 E2 07 1B 83 0B 53 55 50 45 52 42 4C 4F 4F 4D
00 84 03 45 54 48 A0 04 53 45 45 44 A1 2A 30 78
34 45 37 42 64 38 38 45 33 39 39 36 66 34 38 45
32 61 32 34 44 31 35 45 33 37 63 41 34 43 30 32
42 34 44 31 33 34 64 32 A2 01 12 86 40 31 AE DE
CF 5E AE 50 1D 2E EB 07 E5 5C 6F 1B DE FC D3 DC
BE EA 9A EA C7 95 96 4B 60 EA DA BB 11 AB 92 62
F2 D0 14 E8 29 E5 DD 3C 32 20 45 3C 1C 68 D0 D1
19 27 77 1B 1F 72 7C 21 52 86 C8 50 31 03 41 04
1E 2C 7E 19 3A C0 D9 25 ED AC 5E 7B 97 6E 82 0E
ED D7 23 E4 02 C4 14 43 19 17 16 2D 44 25 FF F5
93 07 14 12 3F AB D5 83 D5 C0 9E 5A C0 40 5F 16
17 9F 47 4D 41 8C B2 32 F6 1C 9E 6B 57 3B 32 E4
60 41 04 ED CE A7 82 A6 4E 2A F1 47 64 E6 EC C8
D1 37 2B 3E 65 59 5D 6F 8F BD 8B 08 08 4E 7E A3
7A C4 D3 F3 11 A6 D7 CB 9E 4F 61 BC 88 F4 C1 1C
02 3A A7 46 7D C8 00 46 03 3F 9D C5 A2 D0 34 BC
9A B2 35 08 04 00 0F 42 40 62 04 00 0F 42 40 63
04 00 00 00 00 16 10 0D 03 78 C4 3C 19 48 33 84
53 93 E8 CD 5D D1 B4 17 10 AC 9C 63 93 18 F2 EB
A8 F9 F7 B7 91 0B DE DD F7 61 40 F8 F9 EE 41 43
ED 74 C7 EB 8E A4 E7 F9 21 D1 D6 B8 D3 3E C0 BD
56 C3 5C 02 8F C0 21 C5 D5 B5 7D 3A EE 57 CC 93
74 CB F6 6E B3 B4 AD 54 0E ED 89 C1 86 5A 61 5D
99 60 24 1A CB 07 75 6A D4 18 9A 0F 01 00
```

10. Appendix B – CRC-A

```
byte[] calculateCRC16(byte[] bytes) {
    byte chBlock;
    int wCRC = 0x6363;
    int i = 0;
    do {
       chBlock = bytes[i++];
       chBlock ^= (byte) (wCRC & 0x00FF);
       chBlock = (byte) (chBlock ^ (chBlock << 4));
       wCRC = ((wCRC >> 8) ^ ((chBlock & 0xFF) << 8) & 0xFFFF) ^ (((chBlock & 0xFF) << 3) &
0xFFFF) ^ (((chBlock & 0xFF) >> 4) & 0xFFFF);
     } while (i < bytes.length);
    return new byte[]{(byte) (wCRC & 0xFF), (byte) ((wCRC & 0xFFFF) >> 8)};
  }
```

11. Appendix C – Verification of Luhn code

```
boolean checkCardID( String sCardID ) {
int sum = 0;
int length = sCardID.length();

for (int i = 0; i < length; i++) {
 // get digits in reverse order
 int digit;
 char cDigit = sCardID.charAt(length - i - 1);
 if (cDigit >= '0' && cDigit <= '9') {
  digit = cDigit - '0';
 } else {
  digit = cDigit - 'A';
 }

 // every 2nd number multiply with 2
 if (i % 2 == 1) {
  digit *= 2;
 }
  sum += digit > 9 ? digit - 9 : digit;
 }
 return sum % 10 == 0;
}
```

12. Appendix D - Examples

12.1 Read command

Request

>> 00F2000022102091B4D142823F7D20C5F08DF69122DE43F35F057A988D9619F6D3138485C9A203

CLA	*INS*	*P1*	*P2*	*Lc*	*Payload*		
					Tag	Length	Value
00	F2	00	00	22	10 (*PIN1*)	20	91 B4 D1 42 82 3F 7D 20 C5 F0 8D F6 91 22 DE 43 F3 5F 05 7A 98 8D 96 19 F6 D3 13 84 85 C9 A2 03

Response

>> 0108FF0000000000011120 0B534D41525420434153480020102800A312E3238642053444B000
341044CB1004B43B407419E29A8FFDB64D4E54B623CEB37F3C2037B3ED6F38EEE0C1F2E5AB5D015
DF78FE15EFA5327F59A24C059C999AFC1D3F2A8DDEEE16467CA75F0A027E310C5E8102FFFF820407
E2071B830B54414E47454D2053444B008403425443864 05D7FFCE7446DAA9084595F383E712A63B2
AC4CF7BDE7673F05D6FC629F0D3E0F637910B5A675F66B633331630AEFB614345AF05208DEECF227
4FF3B44642AC883041045F16BD1D2EAFE463E62A335A09E6B2BBCBD04452526885CB679FC4D27AF
1BD22F553C7DEEFB54FD3D4F361D14E6DC3F11B7D4EA183250A60720EBDF9E110CD26050A736563
703235366B3100804000000064070100090205DC604104B45FF0D628E1B59F7AEFA1D5B45AB9D7C4
7FC090D8B29ACCB515431BDBAD2802DDB3AC5E83A06BD8F13ABB84A465CA3C0FA0B44301F80295
A9B4C5E35D5FDFE5620400000006463040000000 0F0100 9000

Tag	Length	*Value*
01 (*CID*)	08	FF 00 00 00 00 00 01 11
20 (*Manufacturer_ID*)	0B	53 4D 41 52 54 20 43 41 53 48 00 (SMART CASH)
02 (Status)	01	02
80 (Firmware)	0A	31 2E 32 38 64 20 53 44 4B 00 (1.28d SDK)
03 (*CARD_PUBLIC_KEY*)	41	04 4C B1 00 4B 43 B4 07 41 9E 29 A8 FF DB 64 D4 E5 4B 62 3C EB 37 F3 C2 03 7B 3E D6 F3 8E EE 0C 1F 2E 5A B5 D0 15 DF 78 FE 15 EF A5 32 7F 59 A2 4C 05 9C 99 9A FC 1D 3F 2A 8D DE EE 16 46 7C A7 5F
0A (*Settings_Mask*)	02	7E 31

0C (*Card_Data*)		5E	81 02 FF FF 82 04 07 E2 07 1B 83 0B 54 41 4E 47 45 4D 20 53 44 4B 00 84 03 42 54 43 8640 5D 7F FC E7 44 6D AA 90 84 59 5F 38 3E 71 2A 63 B2 AC 4C F7 BD E7 67 3F 05 D6 FC 62 9F 0D 3E 0F 63 79 10 B5 A6 75 F6 6B 63 33 31 63 0A EF B6 14 34 5A F0 52 08 DE EC F2 27 4F F3 B4 46 42 AC 88
Card_Data (Detailed)	81 (Batch)	02	FF FF
	82 (*Manufacture_Date_Time*)	04	07 E2 07 1B
	83 (Issuer_ID)	0B	54 41 4E 47 45 4D 20 53 44 4B 00 (TANGEM SDK)
	84 (Blockchain_ID)	03	42 54 43 (BTC)
	86 (*CID_MANUFACTURER_SIGNATURE*)	40	5D 7F FC E7 44 6D AA 90 84 59 5F 38 3E 71 2A 63 B2 AC 4C F7 BD E7 67 3F 05 D6 FC 62 9F 0D 3E 0F 63 79 10 B5 A6 75 F6 6B 63 33 31 63 0A EF B6 14 34 5A F0 52 08 DE EC F2 27 4F F3 B4 46 42 AC 88
30 (*ISSUER_PUBLIC_KEY*)		41	04 5F 16 BD 1D 2E AF E4 63 E6 2A 33 5A 09 E6 B2 BB CB D0 44 52 52 68 85 CB 67 9F C4 D2 7A F1 BD 22 F5 53 C7 DE EF B5 4F D3 D4 F3 61 D1 4E 6D C3 F1 1B 7D 4E A1 83 25 0A 60 72 0E BD F9 E1 10 CD 26
05 (*Curve_ID*)		0A	73 65 63 70 32 35 36 6B 31 00 (secp256k1)
08 (Max_Signatures)		04	00 00 00 64
07 (*Signing_Method*)		01	00
09 (*Pause_Before_PIN2)*		02	05 DC

60 (*WALLET_PUBLIC_KEY)*	41	04 B4 5F F0 D6 28 E1 B5 9F 7A EF A1 D5 B4 5A B9 D7 C4 7F C0 90 D8 B2 9A CC B5 15 43 1B DB AD 28 02 DD B3 AC 5E 83 A0 6B D8 F1 3A BB 84 A4 65 CA 3C 0F A0 B4 43 01 F8 02 95 A9 B4 C5 E3 5D 5F DF E5
62 (*Wallet_Remaining_Signatures)*	04	00 00 00 64
63 (*Wallet_Signed_Hashes*)	04	00 00 00 00
0F (*Health*)	01	00

13. Appendix E – Private files for SSI and other applications

In firmware version 3.29, instead of `IssuerDataEx`, several files can be written to the card (in 3.29 up to 40 files, up to 48 kb in total).

Moreover, each file can be public or private (PIN2 is required to read a private file).

For now, every file must be signed with the `IssuerDataKey` key and can be protected by the `IssuerDataCounter` counter (similar to the IssuerDataEx), but in the future, new modes will probably appear.

Files index starts from 0, `TAG_Index` = 0x26 (1 byte), when the file is deleted, the numbers are shifted.

`IssuerDataSignature` (`TAG_Issuer_Data_Signature` = 0x33) is stored only in case a single file has been written.
File deletion and privacy settings are protected by PIN2.

After one successful PIN2 check, SecurityDelay is not performed during one session.

13.1 File writing

Writing is generally similar to `IssuerDataEx` command, first file recording is initiated (Mode = 1), then several data with the offset (Mode = 2), then the end of writing (Mode = 3).

The differences are:

- new command `INS_WriteFileData` = 0xD0
- start recording command (Mode = 1) adds a new file (the index of the file being added is returned `TAG_Index` = 0x26)
- when writing a piece of data (Mode = 2) and finishing writing a file (Mode = 3), the file index (`TAG_Index` = 0x26, 1 byte) is also transmitted, which must match the one returned at the beginning of writing.

The procedure for writing a new file should look like this:

1. Start writing the file `INS_WriteFileData` = 0xD0 (`TAG_Mode` = 1, `TAG_Size` = fileSize)
2. Write the file parts one by one, performing `INS_WriteFileData` = 0xD0 (`TAG_Mode` = 2, `TAG_Offset` = fileOffset).
3. Confirm the file entry `INS_WriteFileData` = 0xD0 (`TAG_Mode` = 3)

For now, items 1 and 3 must be confirmed by the `IssuerDataSignature` signature.

```
<< [WriteFileData]
<<     TAG_CardID[0x01:8]: AA12345678900001
<<     TAG_PIN[0x10:32]:
91B4D142823F7D20C5F08DF69122DE43F35F057A988D9619F6D3138485C9A203
<<     TAG_Mode[0x23:1]: 01
<<     TAG_Issuer_Data_Counter[0x35:4]: 00000003
<<     TAG_Size[0x25:2]: 00E4
<<     TAG_Issuer_Data_Signature[0x33:64]:
DE79883FF020A0D6DA2D6A2F996F2C8CC567EAB18A348D4C7F77CCBC98A3B203E4AE29FA62E
CB0B458DD2021FBEA120BF9E52FDBABF56B4E272CC1FC96BE0A79
>> OK: [9000]
>>     TAG_CardID[0x01:8]: AA12345678900001
>>     TAG_Index[0x26:1]: 00

<< [WriteFileData]
<<     TAG_CardID[0x01:8]: AA12345678900001
<<     TAG_PIN[0x10:32]:
91B4D142823F7D20C5F08DF69122DE43F35F057A988D9619F6D3138485C9A203
<<     TAG_Mode[0x23:1]: 02
<<     TAG_Index[0x26:1]: 00
<<     TAG_Offset[0x24:2]: 0000
<<     TAG_Issuer_Data[0x32:228]:
00A000000003098AB8F59CF848433D2173C91611884F2EE335C00A44A2C209CE79A3A8381EF
75DE91A99724B8F647C0F190A408D3AC888BC4C438E47828C033CEE985F7556D6C4BD922E9F
73210FCF4B918A4D5CD190E444F4D6986D1C9C37760396BDAC87D8100D82A4F00663DD4490B
123BFFEADBD28BA9B901650FF34C1BE8C2D17ACEFFED0B52275DA7B16589F0CA3C777A6D780
FBDDB6B91A08F42752CCE278334015B33A77C4DA7F1FF6F2A907AEDE0278100C3D1CE2F2630
3E944BE9D0949A42386E6FCC7B6F10748CEF7AD4DB642994B172EBD2A762A2BC91C969B5933
3DF049
>> OK: [9000]
>>     TAG_CardID[0x01:8]: AA12345678900001

<< [WriteFileData]
<<     TAG_CardID[0x01:8]: AA12345678900001
<<     TAG_PIN[0x10:32]:
91B4D142823F7D20C5F08DF69122DE43F35F057A988D9619F6D3138485C9A203
<<     TAG_Mode[0x23:1]: 03
<<     TAG_Index[0x26:1]: 00
<<     TAG_Issuer_Data_Signature[0x33:64]:
F01D746F84815572BB120530590A647F0B146EBF3CAB894C775B20B4896559AA8F40BC821E2
A9D7EB69F3C1054A1EDABDA75CB56B0C2D5DF4AACE96FE24531A2
>> OK: [9000]
>>     TAG_CardID[0x01:8]: AA12345678900001
```

13.2 File reading

Reading is generally similar to the `IssuerDataEx` command.

The differences are:

- Request:
  - The new command, `INS_GetFileData` = 0xD1
  - Mode (`TAG_Mode` = 0x23) is not used

  - Optionally, the index of the file (`TAG_Index` = 0x26) that should be read is transmitted (by default, the file with index 0). If the file with the specified index can't be read (for example, it's private, and PIN2 has not been passed), then a file with the next index is returned.
  - PIN2 can be optionally passed for the ability to read private files.

- Response:
  - If there is no file with an index equal to or greater than the specified one and that meets the search criteria, then the error `SW_FILE_NOT_FOUND` = 0x6A82 is returned
  - The response contains `TAG_Index` = 0x26 - the index of the returned file
  - When reading the first part of the file (with `TAG_Offset` = 0x0000), `TAG_FileSettings` = 0x27 is additionally returned - two bytes, a mask with file settings. 0x0001 - the file is public, 0x0000 - private
  - `IssuerDataSignature` (`TAG_Issuer_Data_Signature` = 0x33) is stored only if there is only one file on the card.

If there are no files, an empty IssuerData will be returned, to be able to read the IssuerDataCounter.

The procedure for reading all files should look like this:

1. Set `fileIndex` = 0
2. Execute the command `INS_GetFileData` (`TAG_Index` = fileIndex, `TAG_Offset` = fileOffset)
3. If the response is `SW_FILE_NOT_FOUND` - all files have been read, the procedure is over.
4. Take fileIndex = `TAG_Index` and fileSize = `TAG_Size` from the reaponse.
5. Repeat INS_GetFileData (`TAG_Index` = fileIndex, `TAG_Offset` = fileOffset) increasing `fileOffset` until fileSize bytes is read (as an option until the TAG_IssuerDataCounter tag is gottn in the response).
6. Increase the `fileIndex` and go to step 2

```
<< [GetFileData]
<<     TAG_CardID[0x01:8]: AA12345678900001
<<     TAG_PIN[0x10:32]:
91B4D142823F7D20C5F08DF69122DE43F35F057A988D9619F6D3138485C9A203
<<     TAG_Index[0x26:1]: 00
<<     TAG_Offset[0x24:2]: 0000
>> OK: [9000]
>>     TAG_CardID[0x01:8]: AA12345678900001
>>     TAG_Offset[0x24:2]: 0000
>>     TAG_Index[0x26:1]: 00
>>     TAG_Size[0x25:2]: 00E4
>>     TAG_FileSettings[0x27:2]: 0001
>>     TAG_Issuer_Data[0x32:228]:
00A000000003098AB8F59CF848433D2173C91611884F2EE335C00A44A2C209CE79A3A8381EF
75DE91A99724B8F647C0F190A408D3AC888BC4C438E47828C033CEE985F7556D6C4BD922E9F
73210FCF4B918A4D5CD190E444F4D6986D1C9C37760396BDAC87D8100D82A4F00663DD4490B
123BFFEADBD28BA9B901650FF34C1BE8C2D17ACEFFED0B52275DA7B16589F0CA3C777A6D780
FBDDB6B91A08F42752CCE278334015B33A77C4DA7F1FF6F2A907AEDE0278100C3D1CE2F2630
3E944BE9D0949A42386E6FCC7B6F10748CEF7AD4DB642994B172EBD2A762A2BC91C969B5933
3DF049
>>     TAG_Issuer_Data_Signature[0x33:64]:
F01D746F84815572BB120530590A647F0B146EBF3CAB894C775B20B4896559AA8F40BC821E2
A9D7EB69F3C1054A1EDABDA75CB56B0C2D5DF4AACE96FE24531A2
>>     TAG_Issuer_Data_Counter[0x35:4]: 00000003

<< [GetFileData]
<<     TAG_CardID[0x01:8]: AA12345678900001
<<     TAG_PIN[0x10:32]:
91B4D142823F7D20C5F08DF69122DE43F35F057A988D9619F6D3138485C9A203
<<     TAG_Index[0x26:1]: 01
<<     TAG_Offset[0x24:2]: 0000
>> Failed: 6A82 - SW_FILE_NOT_FOUND
```

13.3 File deleting

`INS_WriteFileData` = 0xD0 command

Parameters:

- TAG_CardID - CID
- TAG_PIN - PIN
- TAG_PIN2 - PIN2
- TAG_Mode = 0x23 - 0x05 - delete file
- TAG_Index = 0x26 - index of the file to be deleted

```
<< [WriteFileData]
<<     TAG_CardID[0x01:8]: AA12345678900001
<<     TAG_PIN[0x10:32]:
91B4D142823F7D20C5F08DF69122DE43F35F057A988D9619F6D3138485C9A203
<<     TAG_PIN2[0x11:32]:
2AC9A6746ACA543AF8DFF39894CFE8173AFBA21EB01C6FAE33D52947222855EF
<<     TAG_Mode[0x23:1]: 05
<<     TAG_Index[0x26:1]: 00
Security delay: remaining 0,280000 s
>> OK: [9000]
>>     TAG_CardID[0x01:8]: AA12345678900001
```

13.4 Changing file privacy

`INS_WriteFileData` = 0xD0 command

Parameters:

- TAG_CardID - CID
- TAG_PIN - PIN
- TAG_PIN2 - PIN2
- TAG_Mode = 0x23 - 0x06 - change file privacy
- TAG_Index = 0x26 - index of the file
- TAG_FileSettings = 0x27 - settings mask, 0x0001 - public file, 0x0000 - private

```
<< [WriteFileData]
<<     TAG_CardID[0x01:8]: AA12345678900001
<<     TAG_PIN[0x10:32]:
91B4D142823F7D20C5F08DF69122DE43F35F057A988D9619F6D3138485C9A203
<<     TAG_PIN2[0x11:32]:
2AC9A6746ACA543AF8DFF39894CFE8173AFBA21EB01C6FAE33D52947222855EF
<<     TAG_Mode[0x23:1]: 06
<<     TAG_Index[0x26:1]: 00
<<     TAG_FileSettings[0x27:2]: 0000
Security delay: remaining 0,280000 s
>> OK: [9000]
>>     TAG_CardID[0x01:8]: AA12345678900001
```