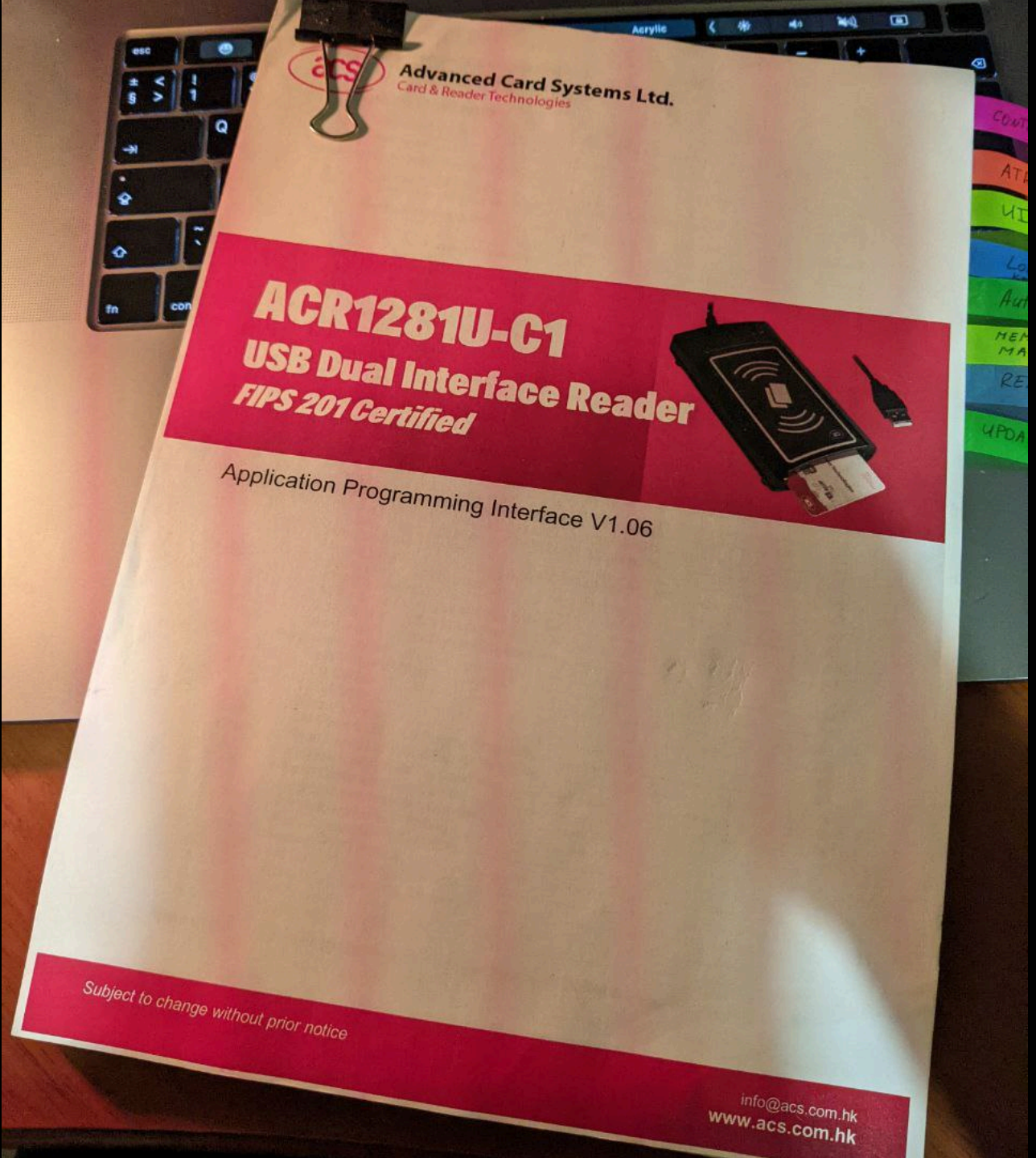
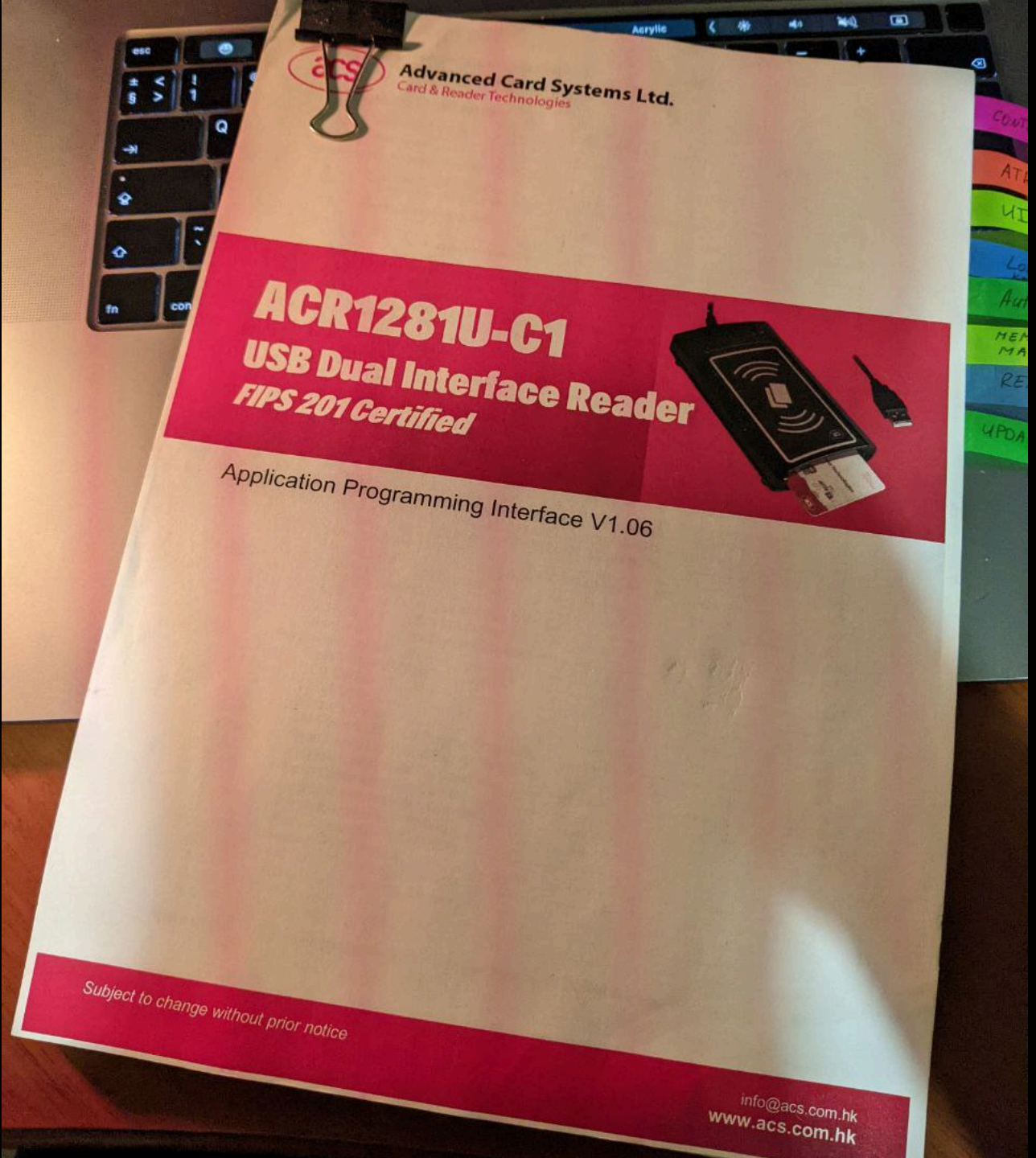
Телефон, NFC и пластиковые карты* - романтика

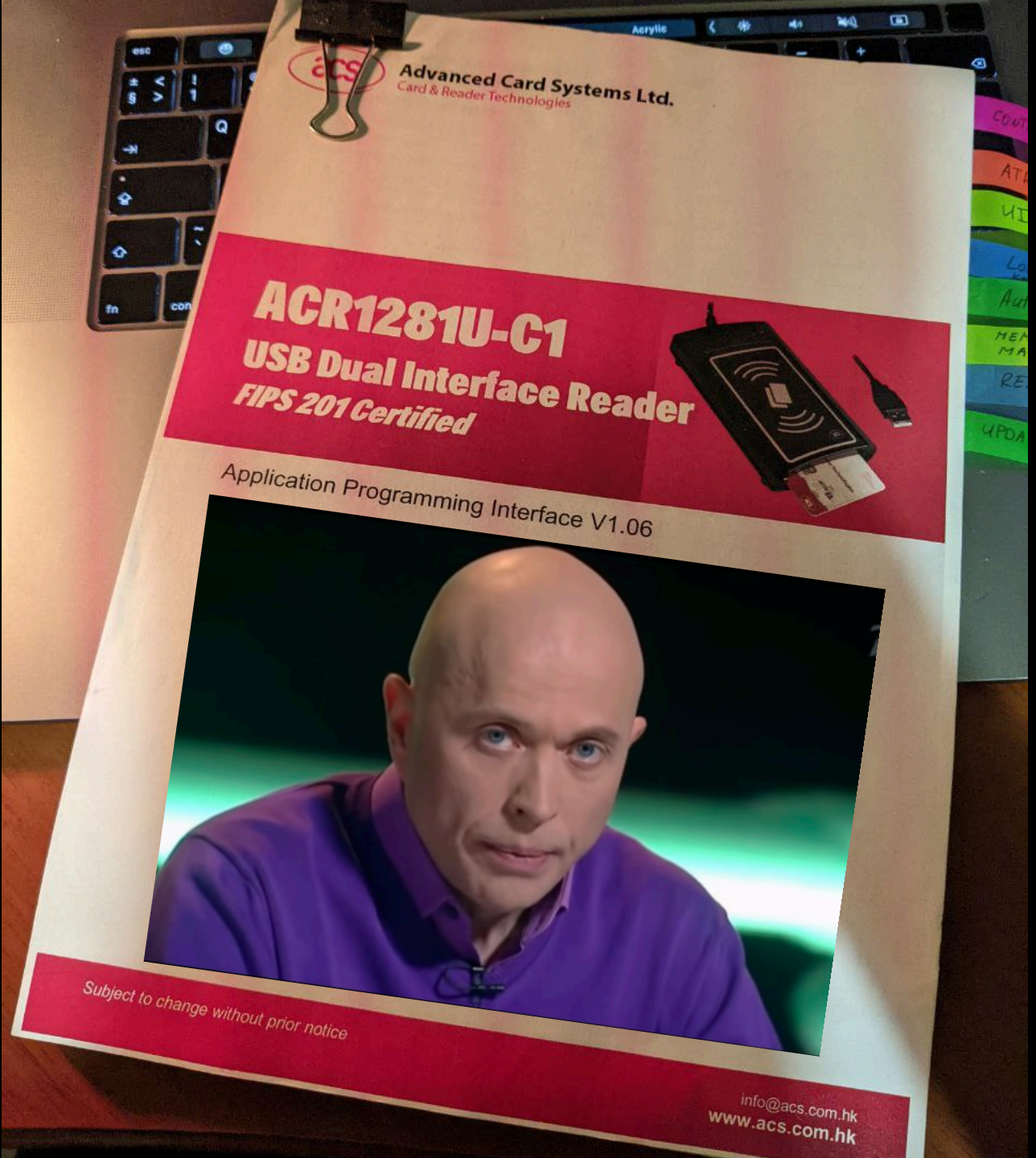
Владислав Кожушко



Внешние считыватели



Внешние считыватели



Внешние считыватели

План

План

- Технология NFC

План

- Технология NFC
- Возможности Android SDK

План

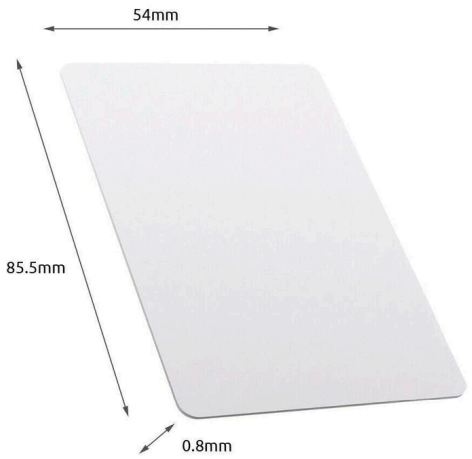
- Технология NFC
- Возможности Android SDK
- Транспортные карты

План

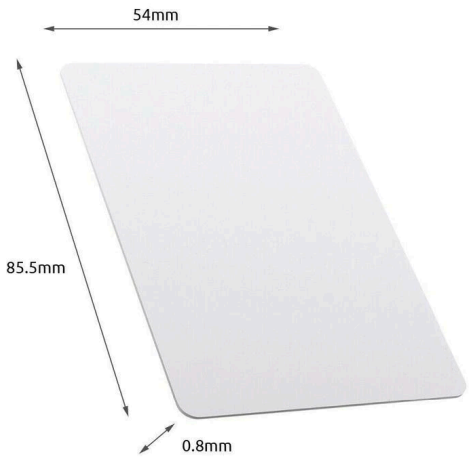
- Технология NFC
- Возможности Android SDK
- Транспортные карты
- Китайские “магические” метки

Технология NFC

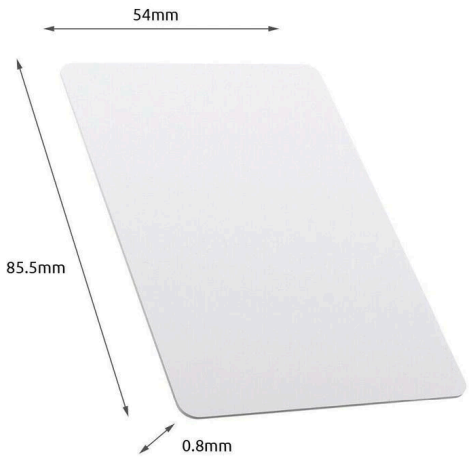
Не только карты



Не только карты

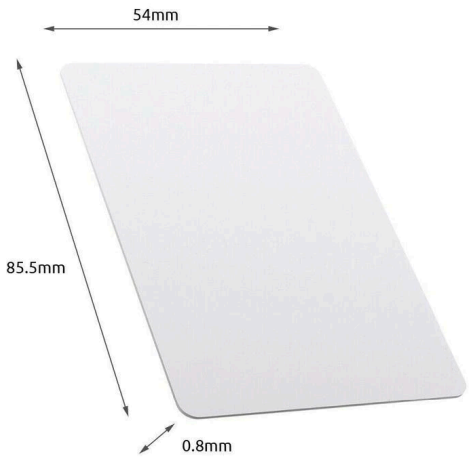


Не только карты

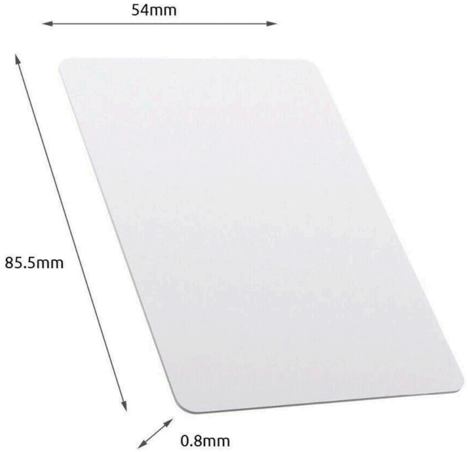


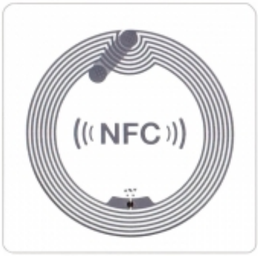
Не только карты

Не только карты

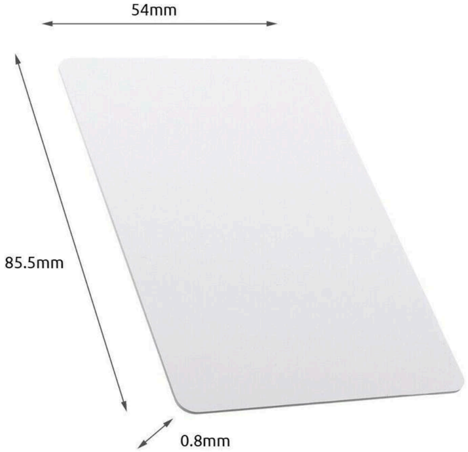


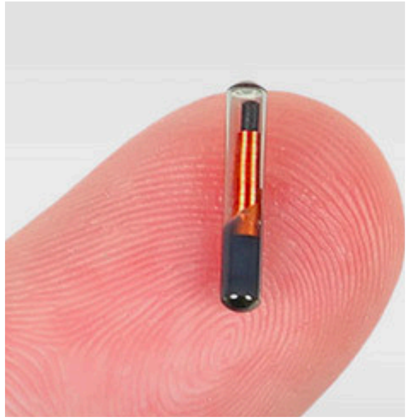
Не только карты



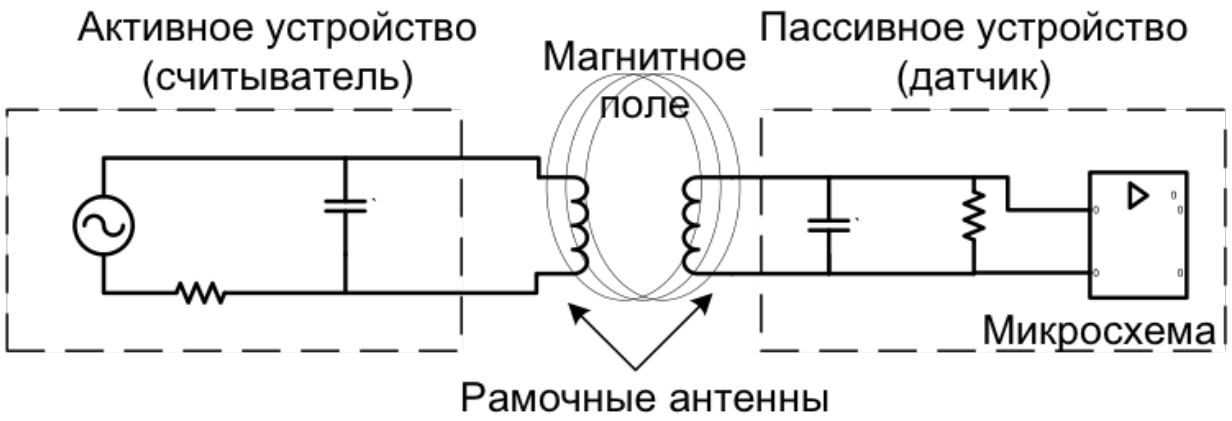


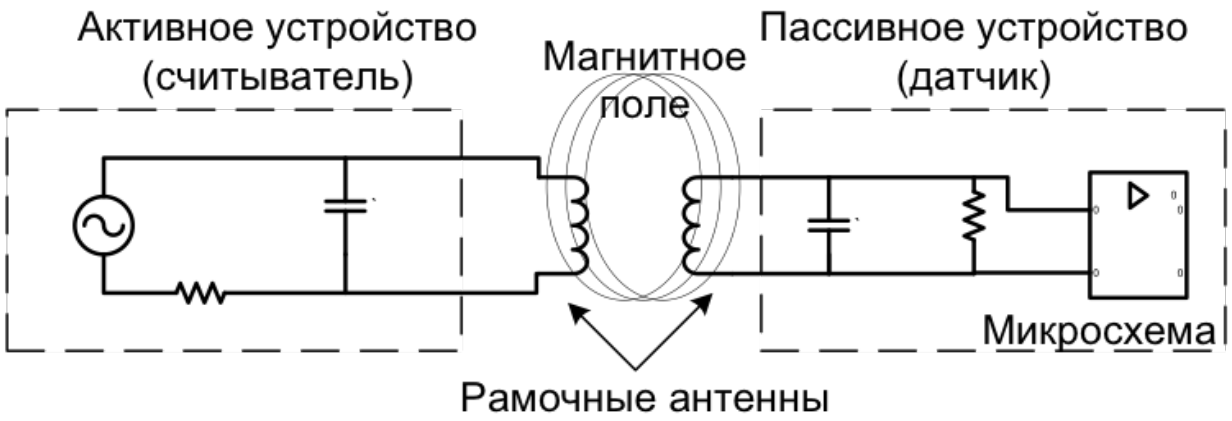
Не только карты





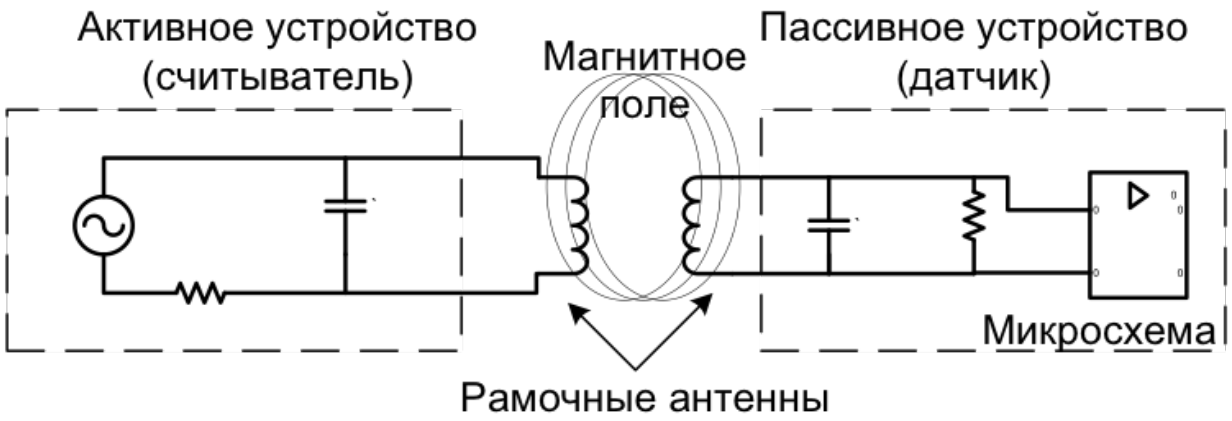
Как работает NFC





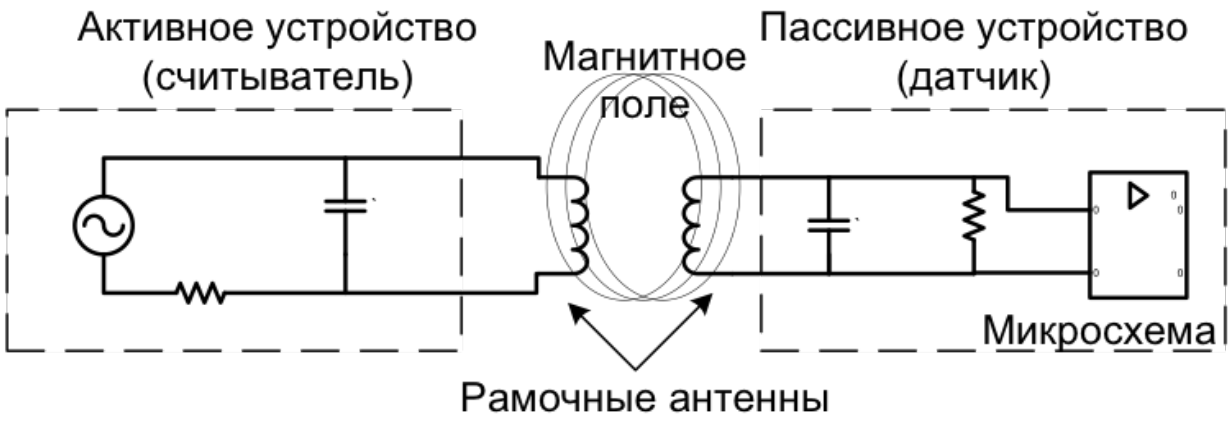
Как работает NFC

1. Мгновенное подключение — 0,1 с



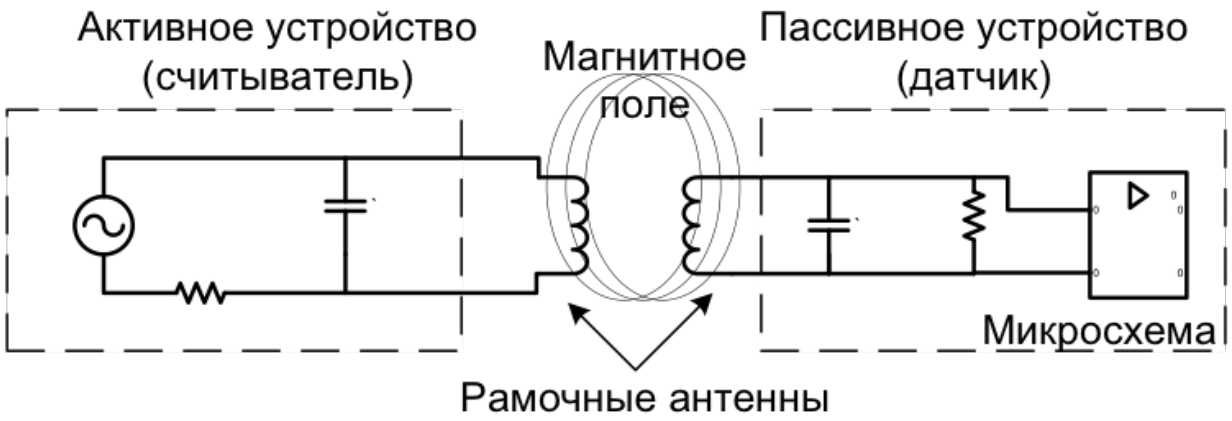
Как работает NFC

1. Мгновенное подключение — 0,1 с
2. Низкое энергопотребление — 15 мА



Как работает NFC

1. Мгновенное подключение — 0,1 с
2. Низкое энергопотребление — 15 мА
3. Малый радиус действия

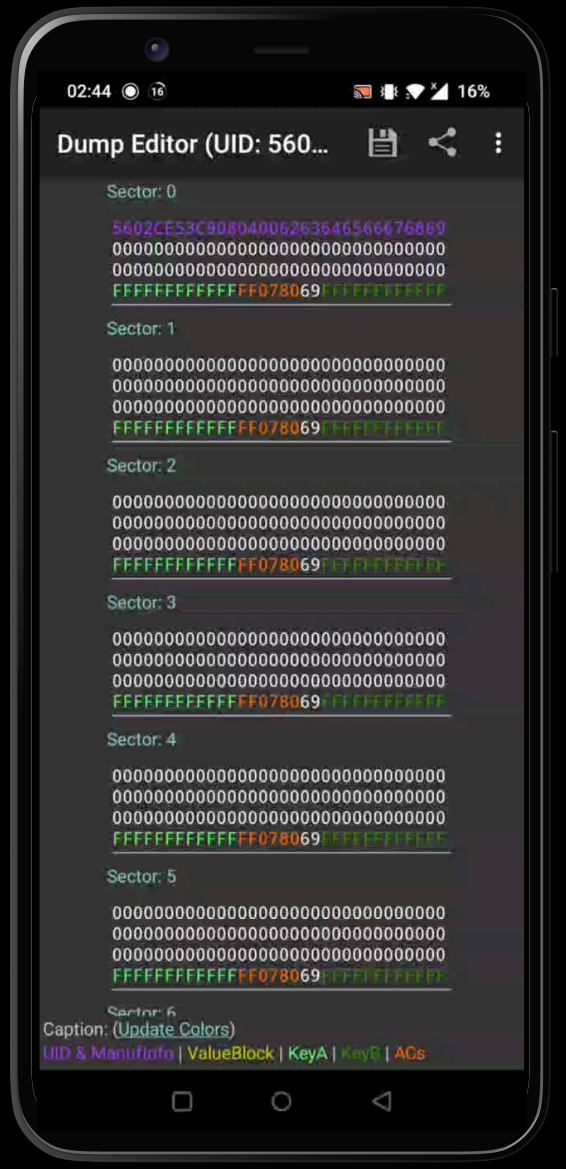


Как работает NFC

1. Мгновенное подключение — 0,1 с
2. Низкое энергопотребление — 15 мА
3. Малый радиус действия
4. Скорость 106 - 848 кбит/с

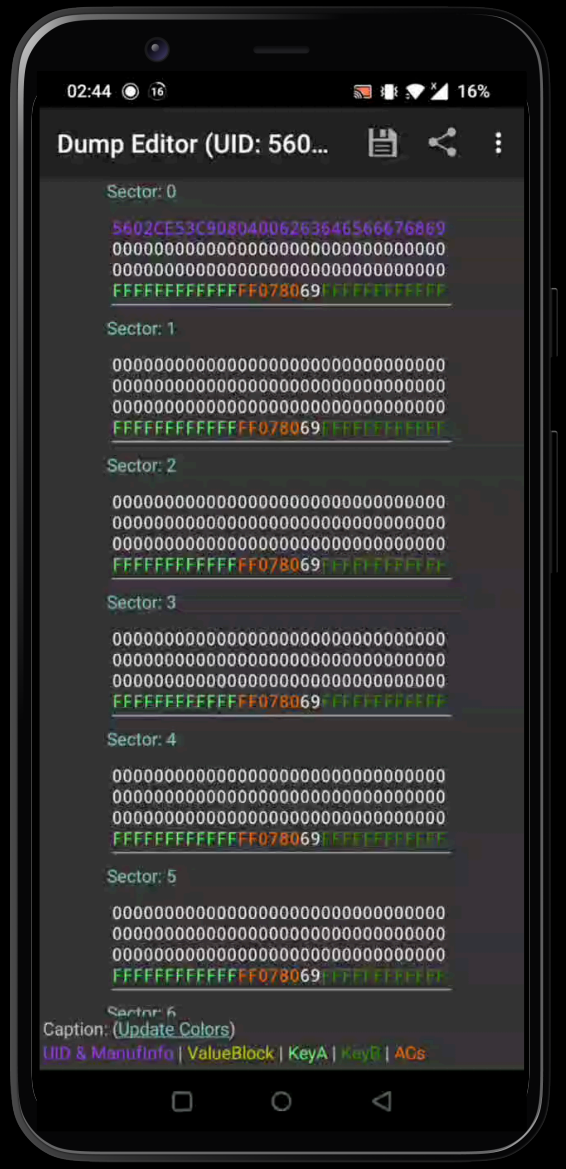
При чем тут RFID?

RFID и NFC


02:44
16%
Dump Editor (UID: 560...
Sector: 0
5602CE53C9080400626364566676869
00000000000000000000000000000000
00000000000000000000000000000000
FFFFFFFFFFFFFF078069FFFFFFFFFFFF
Sector: 1
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
FFFFFFFFFFFFFF078069FFFFFFFFFFFF
Sector: 2
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
FFFFFFFFFFFFFF078069FFFFFFFFFFFF
Sector: 3
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
FFFFFFFFFFFFFF078069FFFFFFFFFFFF
Sector: 4
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
FFFFFFFFFFFFFF078069FFFFFFFFFFFF
Sector: 5
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
FFFFFFFFFFFFFF078069FFFFFFFFFFFF
Caption: (Update Colors)
UID & ManufInfo | ValueBlock | KeyA | KeyB | ACs

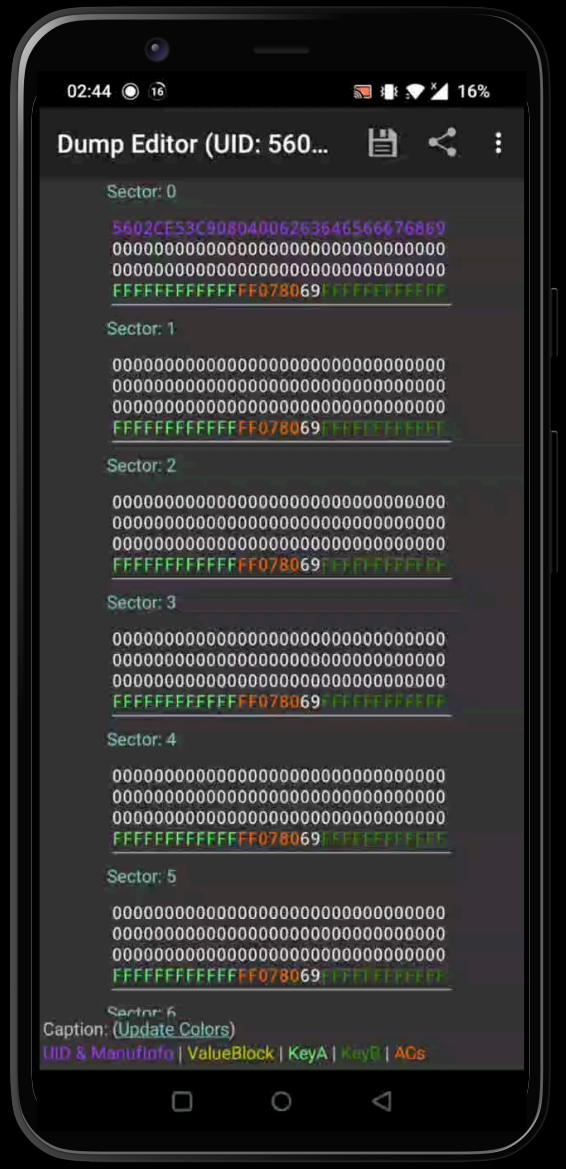
RFID и NFC

- Ближняя (до 20 см)



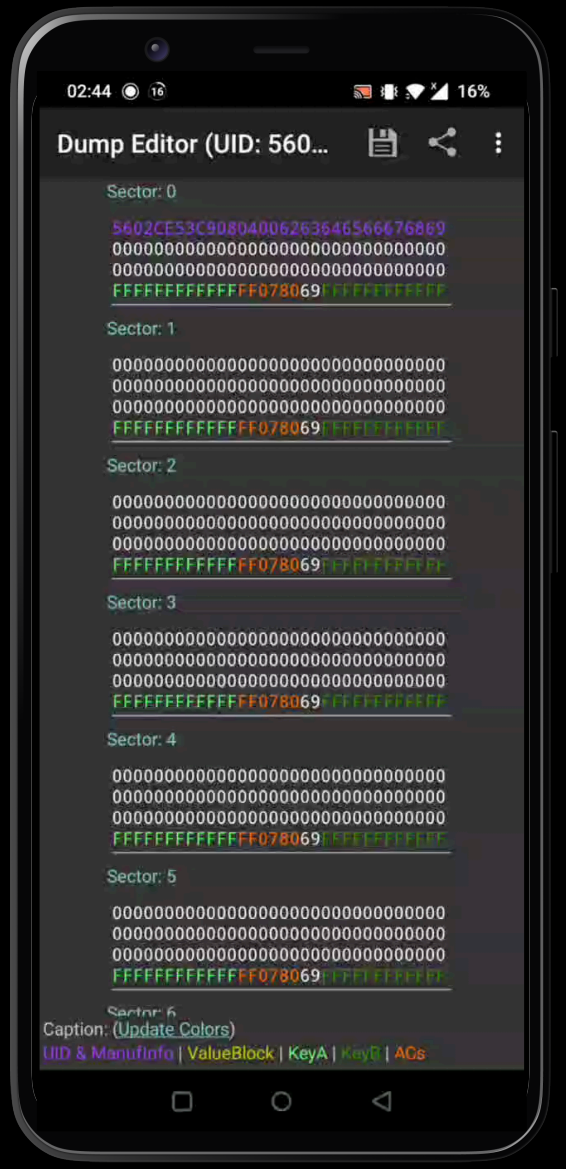
RFID и NFC

- Ближняя (до 20 см)
- Средняя (до 5 м)



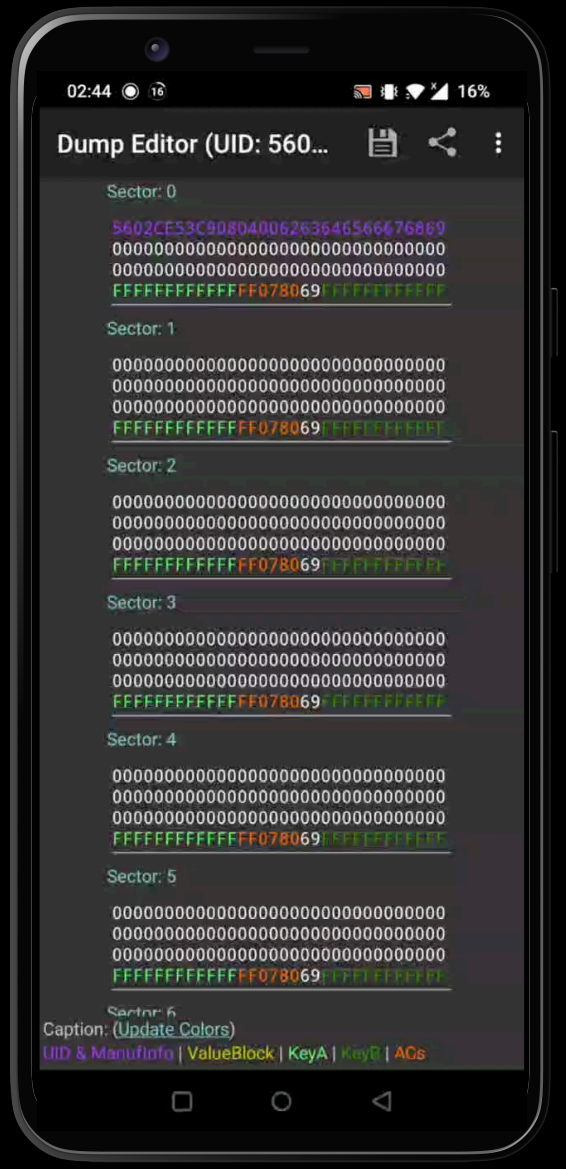
RFID и NFC

- Ближняя (до 20 см)
- Средняя (до 5 м)
- Дальняя (до 300 м)



RFID и NFC

- Ближняя (до 20 см)
- Средняя (до 5 м)
- Дальняя (до 300 м)



Примеры карт

Примеры карт

- EM Marine - 125 КГц

Примеры карт

- EM Marine - 125 КГц
- Mifare (NXP), iClass, iClass SE (HID Corporation) - 13,56 МГц

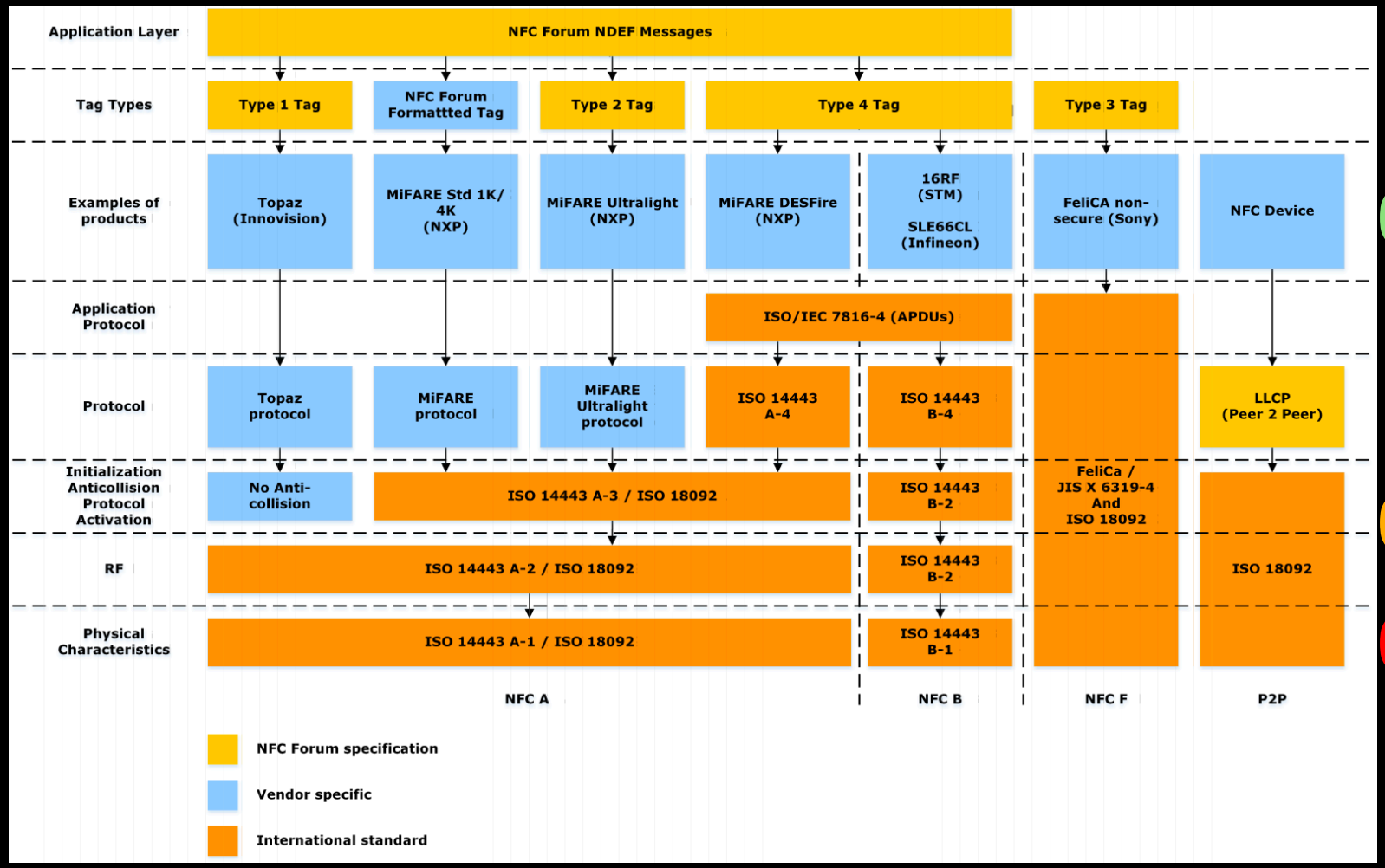
Примеры карт

- EM Marine - 125 КГц
- Mifare (NXP), iClass, iClass SE (HID Corporation) - 13,56 МГц
- Indala (Motorolla) - 860-960 МГц

Примеры карт

- EM Marine - 125 КГц
- Mifare (NXP), iClass, iClass SE (HID Corporation) - 13,56 МГц
- Indala (Motorolla) - 860-960 МГц
- Симкарты - JCOP + Mifare

Протоколы



Прикладной

Канальный

Физический

Носители

Банковские карты (EVM)

Банковские карты (EVM)

- Открытый стандарт EMVCo

Банковские карты (EVM)

- Открытый стандарт EMVCo
- Файловая структура и приложения

Банковские карты (EVM)

- Открытый стандарт EMVCo
- Файловая структура и приложения
- Ключ карты

Банковские карты (EVM)

- Открытый стандарт EMVCo
- Файловая структура и приложения
- Ключ карты
- ОС карты и ключ ОС

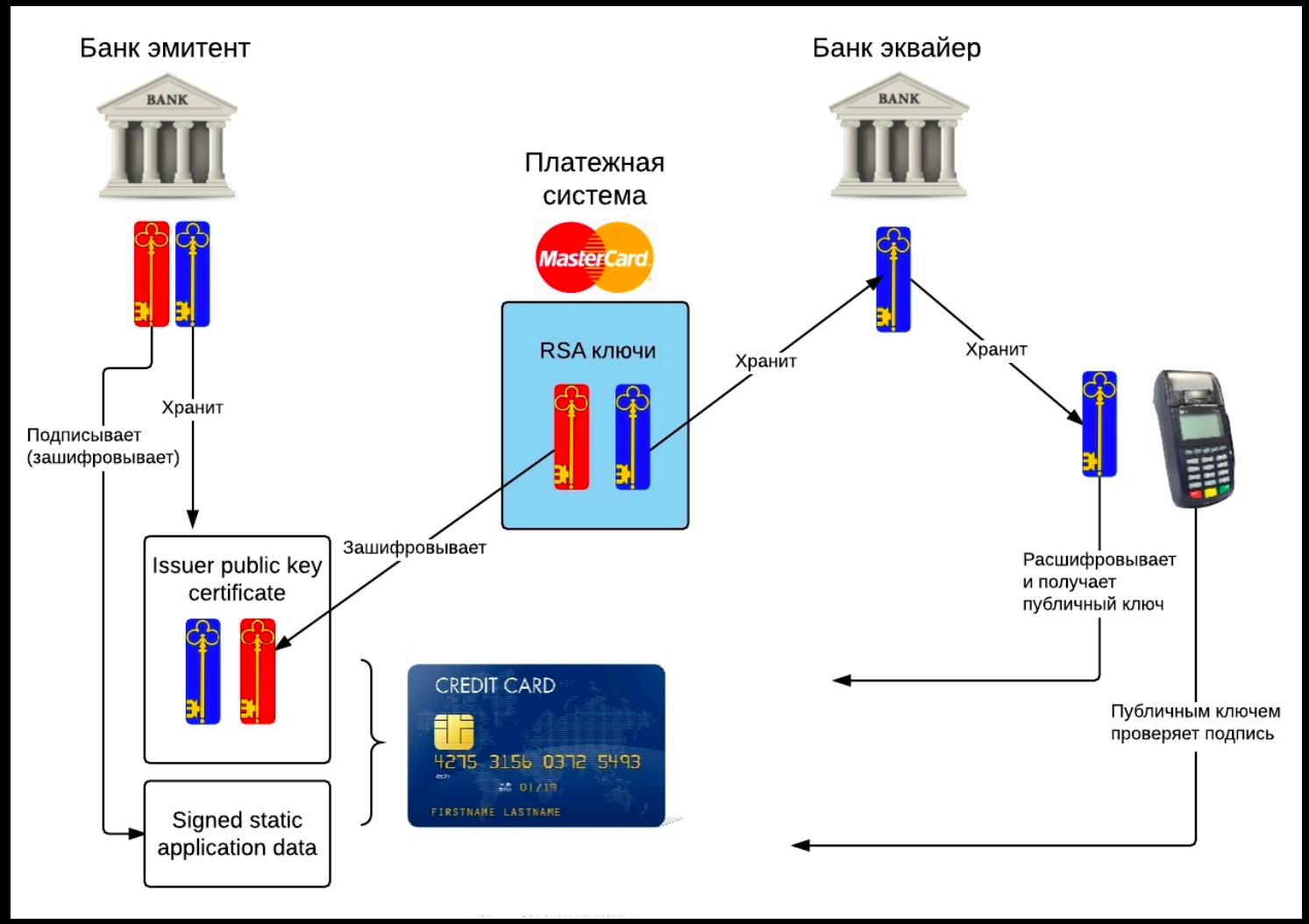
Банковские карты (EVM)

- Открытый стандарт EMVCo
- Файловая структура и приложения
- Ключ карты
- ОС карты и ключ ОС
- Ключ приложения

Банковские карты (EVM)

- Открытый стандарт EMVCo
- Файловая структура и приложения
- Ключ карты
- ОС карты и ключ ОС
- Ключ приложения
- Целостность данных подписывается CVV / CVC кодом

Статическая аутентификация карты



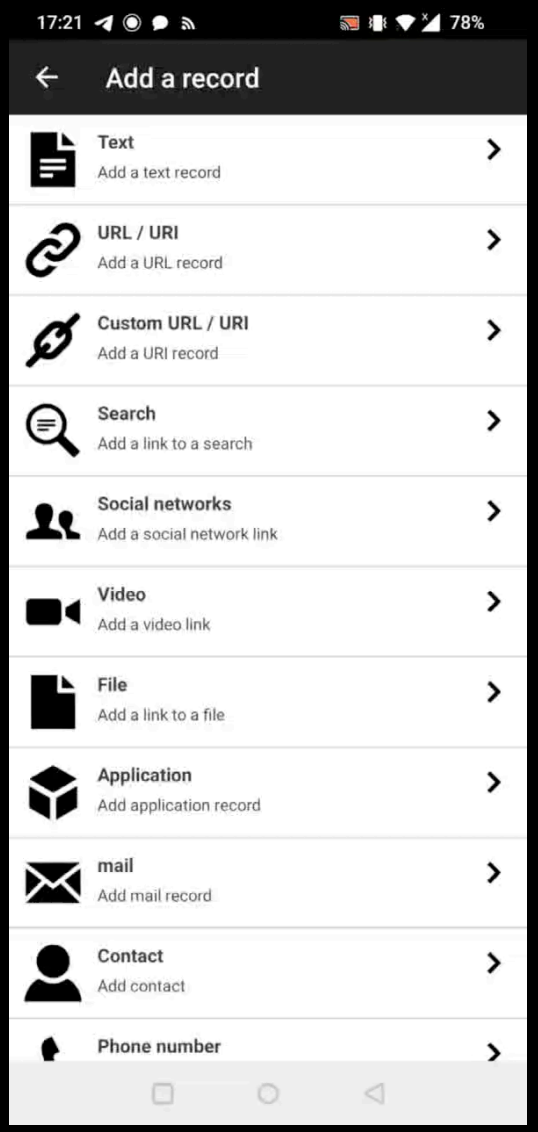
NXP Semiconductors

Ntag 213, 215, 216

Item	Ntag213	Ntag215	Ntag216
Total memory (bytes)	180	540	924
User memory (bytes)	144	504	888
Max URL (chars)	136	488	872
UID (bytes)	7	7	7
ECC signature	✔	✔	✔
32-bit password	✔	✔	✔
UID ASCII mirror	✔	✔	✔
Scan Counter	✔	✔	✔
Counter ASCII mirror	✔	✔	✔
Lockable	✔	✔	✔

Ntag 213, 215, 216

Item	Ntag213	Ntag215	Ntag216
Total memory (bytes)	180	540	924
User memory (bytes)	144	504	888
Max URL (chars)	136	488	872
UID (bytes)	7	7	7
ECC signature	✔	✔	✔
32-bit password	✔	✔	✔
UID ASCII mirror	✔	✔	✔
Scan Counter	✔	✔	✔
Counter ASCII mirror	✔	✔	✔
Lockable	✔	✔	✔



Mifare

- Classic 1k, 2k, 4k
- Ultralight (C, EV1, EV2)
- Plus (S, X)
- DESFire

Mifare Classic 1k/2k/4k

Mifare Classic 1k/2k/4k

- 1к - 16 секторов по 64 б

Mifare Classic 1k/2k/4k

- 1к - 16 секторов по 64 б
- 2к - 32 сектора по 64 б

Mifare Classic 1k/2k/4k

- 1к - 16 секторов по 64 б
- 2к - 32 сектора по 64 б
- 4к - 32 сектора по 64 б + 8 секторов по 256 б

Mifare Classic 1k/2k/4k

- 1к - 16 секторов по 64 б
- 2к - 32 сектора по 64 б
- 4к - 32 сектора по 64 б + 8 секторов по 256 б
- CRYPTO1

Mifare Classic 1k/2k/4k

- 1к - 16 секторов по 64 б
- 2к - 32 сектора по 64 б
- 4к - 32 сектора по 64 б + 8 секторов по 256 б
- CRYPTO1
- Ключи A и B по 6 б

Mifare Classic 1k/2k/4k

- 1к - 16 секторов по 64 б
- 2к - 32 сектора по 64 б
- 4к - 32 сектора по 64 б + 8 секторов по 256 б
- CRYPTO1
- Ключи A и B по 6 б
- Биты доступа

Mifare Classic 1k/2k/4k

- 1к - 16 секторов по 64 б
- 2к - 32 сектора по 64 б
- 4к - 32 сектора по 64 б + 8 секторов по 256 б
- CRYPTO1
- Ключи A и B по 6 б
- Биты доступа
- 0 блок 0 сектора не редактируется* и хранит UID карты

Mifare Classic 1k

	Блок 0	Блок 1	Блок 2	Блок 3
Сектор 0	UID + manufacturer data	Каталог приложений (MAD)		Трейлер сектора
Сектор 1				Трейлер сектора
...				
Сектор 15				Трейлер сектора

Mifare Classic

Mifare classic - атаки

Mifare classic - атаки

- Генератор псевдослучайных чисел (до EV1)

Mifare classic - атаки

- Генератор псевдослучайных чисел (до EV1)
- Повторные аутентификации на другие секторы

Mifare classic - атаки

- Генератор псевдослучайных чисел (до EV1)
- Повторные аутентификации на другие секторы
- MITM

Mifare classic - атаки

- Генератор псевдослучайных чисел (до EV1)
- Повторные аутентификации на другие секторы
- MITM

Mifare Ultralight / Ultralight C / Ultralight EV1

Mifare Ultralight / Ultralight C / Ultralight EV1

- 64 б / 192 б памяти

Mifare Ultralight / Ultralight C / Ultralight EV1

- 64 б / 192 б памяти
- Постраничная структура памяти

Mifare Ultralight / Ultralight C / Ultralight EV1

- 64 б / 192 б памяти
- Постраничная структура памяти
- OTP - 4 б

Mifare Ultralight / Ultralight C / Ultralight EV1

- 64 б / 192 б памяти
- Постраничная структура памяти
- OTP - 4 б
- Постраничная защита записи

Mifare Ultralight / Ultralight C / Ultralight EV1

- 64 б / 192 б памяти
- Постраничная структура памяти
- OTP - 4 б
- Постраничная защита записи
- 3DES-аутентификация (Ultralight C)

Mifare Ultralight / Ultralight C / Ultralight EV1

- 64 б / 192 б памяти
- Постраничная структура памяти
- OTP - 4 б
- Постраничная защита записи
- 3DES-аутентификация (Ultralight C)
- 3 24-bit счетчика (Ultralight EV1)

Mifare Ultralight

Byte Number	0	1	2	3	Page
Serial Number	SN0	SN1	SN2	BCC0	0
Serial Number	SN3	SN4	SN5	SN6	1
Internal / Lock	BCC1	Internal	Lock0	Lock1	2
OTP	OTP0	OTP1	OTP2	OTP3	3
Data read/write	Data0	Data1	Data2	Data3	4
Data read/write	Data4	Data5	Data6	Data7	5
Data read/write	Data8	Data9	Data10	Data11	6
Data read/write	Data12	Data13	Data14	Data15	7
Data read/write	Data16	Data17	Data18	Data19	8
Data read/write	Data20	Data21	Data22	Data23	9
Data read/write	Data24	Data25	Data26	Data27	10
Data read/write	Data28	Data29	Data30	Data31	11
Data read/write	Data32	Data33	Data34	Data35	12
Data read/write	Data36	Data37	Data38	Data39	13
Data read/write	Data40	Data41	Data42	Data43	14
Data read/write	Data44	Data45	Data46	Data47	15

7 Byte Serial Number (UID)

Read-Only Lock

Write-Once Area

User Data Area

Mifare Plus (S, X)

Mifare Plus (S, X)

- 32 сектора aka Mifare Classic 2k

Mifare Plus (S, X)

- 32 сектора aka Mifare Classic 2k
- 3 уровня безопасности (SL1, SL2, SL3)

Mifare Plus (S, X)

- 32 сектора aka Mifare Classic 2k
- 3 уровня безопасности (SL1, SL2, SL3)
- Аутентификация по CRYPTO1 или AES128

Mifare Plus (S, X)

- 32 сектора aka Mifare Classic 2k
- 3 уровня безопасности (SL1, SL2, SL3)
- Аутентификация по CRYPTO1 или AES128
- EEPROM - 2кб

Mifare Plus (S, X)

- 32 сектора aka Mifare Classic 2k
- 3 уровня безопасности (SL1, SL2, SL3)
- Аутентификация по CRYPTO1 или AES128
- EEPROM - 2кб
- Скорость 106 - 848 кбит/с

Что могут смартфоны?

Что могут смартфоны?

Что могут смартфоны?

- Чтение и запись меток
- Эмуляция меток
- Одноранговый обмен (Android beam)

Android SDK

Android SDK

Android SDK

- Mifare Classic*

Android SDK

- Mifare Classic*
- Mifare Ultralight

Android SDK

- Mifare Classic*
- Mifare Ultralight
- NfcA - NFC-A (ISO 14443-3A)

Android SDK

- Mifare Classic*
- Mifare Ultralight
- NfcA - NFC-A (ISO 14443-3A)
- NfcB - NFC-B (ISO 14443-3B)

Android SDK

- Mifare Classic*
- Mifare Ultralight
- NfcA - NFC-A (ISO 14443-3A)
- NfcB - NFC-B (ISO 14443-3B)
- IsoDep - ISO-DEP (ISO 14443-4)

Android SDK

- Mifare Classic*
- Mifare Ultralight
- NfcA - NFC-A (ISO 14443-3A)
- NfcB - NFC-B (ISO 14443-3B)
- IsoDep - ISO-DEP (ISO 14443-4)
- NfcF - NFC-F (JIS 6319-4)

Android SDK

- Mifare Classic*
- Mifare Ultralight
- NfcA - NFC-A (ISO 14443-3A)
- NfcB - NFC-B (ISO 14443-3B)
- IsoDep - ISO-DEP (ISO 14443-4)
- NfcF - NFC-F (JIS 6319-4)
- NfcV - NFC-V (ISO 15693)

Обработка тегов

Обработка тегов

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android">

</manifest>
```

Обработка тегов

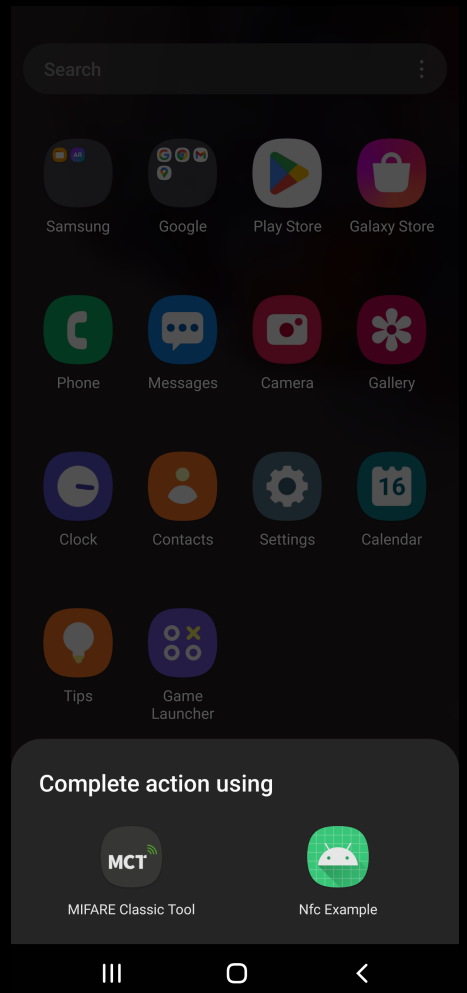
```
<manifest xmlns:android="http://schemas.android.com/apk/res/android">

    <uses-permission android:name="android.permission.NFC" />

    <uses-feature
        android:name="android.hardware.nfc"
        android:required="true" />

</manifest>
```

Обработка тегов

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android">

    <uses-permission android:name="android.permission.NFC" />

    <uses-feature
        android:name="android.hardware.nfc"
        android:required="true" />

    <application
        ...
        >
        <activity
            android:name=".MainActivity"
            android:exported="true">
            ...









        </activity>
    </application>

</manifest>
```

Обработка тегов

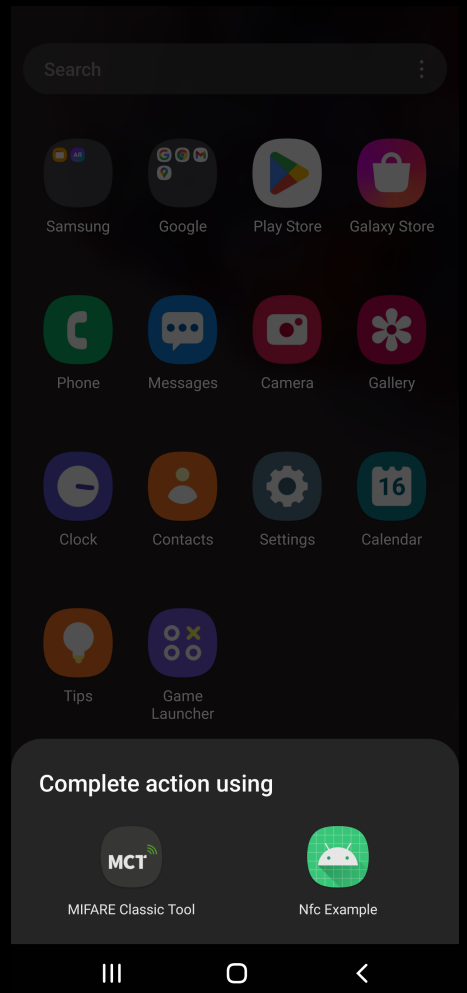
```
<manifest xmlns:android="http://schemas.android.com/apk/res/android">

    <uses-permission android:name="android.permission.NFC" />

    <uses-feature
        android:name="android.hardware.nfc"
        android:required="true" />

    <application
        ...
        >
        <activity
            android:name=".MainActivity"
            android:exported="true">
            ...
```




```
        </activity>
    </application>

</manifest>
```

Обработка тегов

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android">

    <uses-permission android:name="android.permission.NFC" />

    <uses-feature
        android:name="android.hardware.nfc"
        android:required="true" />

    <application
        ...
        >
        <activity
            android:name=".MainActivity"
            android:exported="true">
            ...

            <intent-filter>
                <action android:name="android.nfc.action.TECH_DISCOVERED" />
            </intent-filter>

            <meta-data
                android:name="android.nfc.action.TECH_DISCOVERED"
                android:resource="@xml/nfc_tech" />

        </activity>
    </application>

</manifest>
```



Список технологий

```
<resources xmlns:xliff="urn:oasis:names:tc:xliff:document:1.2">



</resources>
```

Список технологий

```
<resources xmlns:xliff="urn:oasis:names:tc:xliff:document:1.2">
    <tech-list>
        <tech>android.nfc.tech.NfcA</tech>
    </tech-list>



</resources>
```

Список технологий

```xml
<resources xmlns:xliff="urn:oasis:names:tc:xliff:document:1.2">
    <tech-list>
        <tech>android.nfc.tech.NfcA</tech>
    </tech-list>
    <tech-list>
        <tech>android.nfc.tech.NfcA</tech>
        <tech>android.nfc.tech.MifareClassic</tech>
    </tech-list>
</resources>
```

Обработка тегов - способ 1

```
class MainActivity : AppCompatActivity() {

}
```

Обработка тегов - способ 1

```
class MainActivity : AppCompatActivity() {

    private val nfcAdapter: NfcAdapter? by lazy { NfcAdapter.getDefaultAdapter(this) }
    private val nfcHandler = NfcHandler()

}
```

Обработка тегов - способ 1

```
class MainActivity : AppCompatActivity() {

    private val nfcAdapter: NfcAdapter? by lazy { NfcAdapter.getDefaultAdapter(this) }
    private val nfcHandler = NfcHandler()

    override fun onCreate(savedInstanceState: Bundle?) {
        ...
        maybeHandleTag(intent)
    }

    override fun onNewIntent(intent: Intent) {
        super.onNewIntent(intent)

        maybeHandleTag(intent)
    }

}
```

Обработка тегов - способ 1

```
class MainActivity : AppCompatActivity() {

    private val nfcAdapter: NfcAdapter? by lazy { NfcAdapter.getDefaultAdapter(this) }
    private val nfcHandler = NfcHandler()

    override fun onCreate(savedInstanceState: Bundle?) {
        ...
        maybeHandleTag(intent)
    }

    override fun onNewIntent(intent: Intent) {
        super.onNewIntent(intent)

        maybeHandleTag(intent)
    }

    private fun maybeHandleTag(intent: Intent) {
        if (intent.action == NfcAdapter.ACTION_TAG_DISCOVERED ||
            intent.action == NfcAdapter.ACTION_TECH_DISCOVERED
        ) {
            val tag = intent.getParcelableExtra<Tag>(NfcAdapter.EXTRA_TAG)
            tag?.let(::onNewTag)
        }
    }

    private fun onNewTag(tag: Tag) {
        nfcHandler.handleTag(tag)
        Toast.makeText(this, "new tag ${tag.id.toHexString()}", Toast.LENGTH_SHORT)
            .show()
    }
}
```

Обработка тегов - способ 1

```
class MainActivity : AppCompatActivity() {
    private val nfcAdapter: NfcAdapter? by lazy { NfcAdapter.getDefaultAdapter(this) }

}
```

Обработка тегов - способ 1

```
class MainActivity : AppCompatActivity() {

    private val nfcAdapter: NfcAdapter? by lazy { NfcAdapter.getDefaultAdapter(this) }

    override fun onResume() {
        super.onResume()

        nfcAdapter?.let { adapter →
            if (!adapter.isEnabled) {

            } else {
                val intent = Intent(this, javaClass).addFlags(Intent.FLAG_ACTIVITY_SINGLE_TOP)
                adapter.enableForegroundDispatch(
                    this,
                    PendingIntent.getActivity(this, 0, intent, PendingIntent.FLAG_MUTABLE),
                    null,
                    null
                )
            }
        }
    }

}
```

Обработка тегов - способ 1

```
class MainActivity : AppCompatActivity() {

    private val nfcAdapter: NfcAdapter? by lazy { NfcAdapter.getDefaultAdapter(this) }

    override fun onResume() {
        super.onResume()

        nfcAdapter?.let { adapter →
            if (!adapter.isEnabled) {

            } else {
                val intent = Intent(this, javaClass).addFlags(Intent.FLAG_ACTIVITY_SINGLE_TOP)
                adapter.enableForegroundDispatch(
                    this,
                    PendingIntent.getActivity(this, 0, intent, PendingIntent.FLAG_MUTABLE),
                    null,
                    null
                )
            }
        }
    }

    override fun onPause() {
        super.onPause()

        nfcAdapter?.run {
            disableForegroundDispatch(this@MainActivity)
        }
    }
}
```

Обработка тегов - способ 2

```
class MainActivity : AppCompatActivity() {

}
```

Обработка тегов - способ 2

```
class MainActivity : AppCompatActivity() {

    private val nfcAdapter: NfcAdapter? by lazy { NfcAdapter.getDefaultAdapter(this) }
    private val nfcHandler = NfcHandler()
    private val nfcCallback = NfcAdapter.ReaderCallback { onNewTag(it) }
```



```
}
```

Обработка тегов - способ 2

```
class MainActivity : AppCompatActivity() {

    private val nfcAdapter: NfcAdapter? by lazy { NfcAdapter.getDefaultAdapter(this) }
    private val nfcHandler = NfcHandler()
    private val nfcCallback = NfcAdapter.ReaderCallback { onNewTag(it) }

    override fun onResume() {
        super.onResume()

        nfcAdapter?.let { adapter →
            adapter.enableReaderMode(this, nfcCallback,
                NfcAdapter.FLAG_READER_NFC_A,
                //or NfcAdapter.FLAG_READER_NO_PLATFORM_SOUNDS,
                null
            )
        }
    }

}
```

Обработка тегов - способ 2

```
class MainActivity : AppCompatActivity() {

    private val nfcAdapter: NfcAdapter? by lazy { NfcAdapter.getDefaultAdapter(this) }
    private val nfcHandler = NfcHandler()
    private val nfcCallback = NfcAdapter.ReaderCallback { onNewTag(it) }

    override fun onResume() {
        super.onResume()

        nfcAdapter?.let { adapter →
            adapter.enableReaderMode(this, nfcCallback,
                NfcAdapter.FLAG_READER_NFC_A,
                //or NfcAdapter.FLAG_READER_NO_PLATFORM_SOUNDS,
                null
            )
        }
    }

    override fun onPause() {
        super.onPause()

        nfcAdapter?.disableReaderMode(this)
    }



}
```

Обработка тегов - способ 2

```
class MainActivity : AppCompatActivity() {

    private val nfcAdapter: NfcAdapter? by lazy { NfcAdapter.getDefaultAdapter(this) }
    private val nfcHandler = NfcHandler()
    private val nfcCallback = NfcAdapter.ReaderCallback { onNewTag(it) }

    override fun onResume() {
        super.onResume()

        nfcAdapter?.let { adapter →
            adapter.enableReaderMode(this, nfcCallback,
                NfcAdapter.FLAG_READER_NFC_A,
                //or NfcAdapter.FLAG_READER_NO_PLATFORM_SOUNDS,
                null
            )
        }
    }

    override fun onPause() {
        super.onPause()

        nfcAdapter?.disableReaderMode(this)
    }

    private fun onNewTag(tag: Tag) {
        nfcHandler.handleTag(tag)
        runOnUiThread {
            Toast.makeText(this, "new tag ${tag.id.toHexString()}", Toast.LENGTH_SHORT)
                .show()
        }
    }
}
```

Обработка технологии

```
enum class NfcTags(val value: String) {
    MIFARE_CLASSIC("android.nfc.tech.MifareClassic"),
    MIFARE_ULTRALIGHT("android.nfc.tech.MifareUltralight"),
    NFC_A("android.nfc.tech.NfcA"),
}

class NfcHandler {

}
```

Обработка технологии

```
enum class NfcTags(val value: String) {
    MIFARE_CLASSIC("android.nfc.tech.MifareClassic"),
    MIFARE_ULTRALIGHT("android.nfc.tech.MifareUltralight"),
    NFC_A("android.nfc.tech.NfcA"),
}

class NfcHandler {

    fun handleTag(tag: Tag) {
        tag.techList.forEach { tech →
            when (tech) {



            }
        }
    }




}
```

Обработка технологии

```
enum class NfcTags(val value: String) {
    MIFARE_CLASSIC("android.nfc.tech.MifareClassic"),
    MIFARE_ULTRALIGHT("android.nfc.tech.MifareUltralight"),
    NFC_A("android.nfc.tech.NfcA"),
}

class NfcHandler {

    fun handleTag(tag: Tag) {
        tag.techList.forEach { tech →
            when (tech) {
                NfcTags.MIFARE_ULTRALIGHT.value → handleMifareUltralight(MifareUltralight.get(tag))


            }
        }
    }




}
```

Обработка технологии

```
enum class NfcTags(val value: String) {
    MIFARE_CLASSIC("android.nfc.tech.MifareClassic"),
    MIFARE_ULTRALIGHT("android.nfc.tech.MifareUltralight"),
    NFC_A("android.nfc.tech.NfcA"),
}

class NfcHandler {

    fun handleTag(tag: Tag) {
        tag.techList.forEach { tech →
            when (tech) {
                NfcTags.MIFARE_ULTRALIGHT.value → handleMifareUltralight(MifareUltralight.get(tag))
                NfcTags.MIFARE_CLASSIC.value → handleMifareClassic(MifareClassic.get(tag))

            }
        }
    }




}
```

Обработка технологии

```
enum class NfcTags(val value: String) {
    MIFARE_CLASSIC("android.nfc.tech.MifareClassic"),
    MIFARE_ULTRALIGHT("android.nfc.tech.MifareUltralight"),
    NFC_A("android.nfc.tech.NfcA"),
}

class NfcHandler {

    fun handleTag(tag: Tag) {
        tag.techList.forEach { tech →
            when (tech) {
                NfcTags.MIFARE_ULTRALIGHT.value → handleMifareUltralight(MifareUltralight.get(tag))
                NfcTags.MIFARE_CLASSIC.value → handleMifareClassic(MifareClassic.get(tag))
                NfcTags.NFC_A.value → handleNfcA(NfcA.get(tag))
            }
        }
    }




}
```

Обработка технологии

```
enum class NfcTags(val value: String) {
    MIFARE_CLASSIC("android.nfc.tech.MifareClassic"),
    MIFARE_ULTRALIGHT("android.nfc.tech.MifareUltralight"),
    NFC_A("android.nfc.tech.NfcA"),
}

class NfcHandler {

    fun handleTag(tag: Tag) {
        tag.techList.forEach { tech →
            when (tech) {
                NfcTags.MIFARE_ULTRALIGHT.value → handleMifareUltralight(MifareUltralight.get(tag))
                NfcTags.MIFARE_CLASSIC.value → handleMifareClassic(MifareClassic.get(tag))
                NfcTags.NFC_A.value → handleNfcA(NfcA.get(tag))
            }
        }
    }

    private fun handleMifareUltralight(ultralight: MifareUltralight) {}

    private fun handleMifareClassic(classic: MifareClassic) {}

    private fun handleNfcA(nfcA: NfcA) {}
}
```

Чтение Mifare Classic

```
private fun handleMifareClassic(classic: MifareClassic) {
    classic.use { tag →
        try {
            tag.connect()
            readClassic(tag, keys)
        } catch (e: TagLostException) {

        } catch (e: IOException) {

        }
    }
}
```

Чтение Mifare Classic

```
private fun readClassic(tag: MifareClassic, keys: Set<ByteArray>) {

}
```

Чтение Mifare Classic

```
private fun readClassic(tag: MifareClassic, keys: Set<ByteArray>) {
    for (i in 0 until tag.sectorCount) {









    }
}
```

Чтение Mifare Classic

```
private fun readClassic(tag: MifareClassic, keys: Set<ByteArray>) {
    for (i in 0 until tag.sectorCount) {
        val key = keys.firstOrNull { tag.authenticateSectorWithKeyA(i, it) }
        //val key = keys.firstOrNull { tag.authenticateSectorWithKeyB(i, it) }









    }
}
```

Чтение Mifare Classic

```
private fun readClassic(tag: MifareClassic, keys: Set<ByteArray>) {
    for (i in 0 until tag.sectorCount) {
        val key = keys.firstOrNull { tag.authenticateSectorWithKeyA(i, it) }
        //val key = keys.firstOrNull { tag.authenticateSectorWithKeyB(i, it) }
        if (key ≠ null) {





        }
    }
}
```

Чтение Mifare Classic

```
private fun readClassic(tag: MifareClassic, keys: Set<ByteArray>) {
    for (i in 0 until tag.sectorCount) {
        val key = keys.firstOrNull { tag.authenticateSectorWithKeyA(i, it) }
        //val key = keys.firstOrNull { tag.authenticateSectorWithKeyB(i, it) }
        if (key ≠ null) {
            Log.d("Classic", "Sector: $i key = ${key.toHexString()}")
            val blocksCount = tag.getBlockCountInSector(i)



        }
    }
}
```

Чтение Mifare Classic

```
private fun readClassic(tag: MifareClassic, keys: Set<ByteArray>) {
    for (i in 0 until tag.sectorCount) {
        val key = keys.firstOrNull { tag.authenticateSectorWithKeyA(i, it) }
        //val key = keys.firstOrNull { tag.authenticateSectorWithKeyB(i, it) }
        if (key ≠ null) {
            Log.d("Classic", "Sector: $i key = ${key.toHexString()}")
            val blocksCount = tag.getBlockCountInSector(i)
            for (block in 0 until blocksCount) {
                val blockData = tag.readBlock(i * blocksCount + block).toHexString()
                Log.d("Classic", blockData)
            }
        }
    }
}
```

Запись Mifare Classic

```
private fun writeClassic(tag: MifareClassic, keys: Set<ByteArray>, sector: Int, data: ByteArray) {

}
```

Запись Mifare Classic

```
private fun writeClassic(tag: MifareClassic, keys: Set<ByteArray>, sector: Int, data: ByteArray) {
    val key = keys.firstOrNull { tag.authenticateSectorWithKeyA(sector, it) }
    //val key = keys.firstOrNull { tag.authenticateSectorWithKeyB(sector, it) }
    if (key ≠ null) {









    }
}
```

Запись Mifare Classic

```
private fun writeClassic(tag: MifareClassic, keys: Set<ByteArray>, sector: Int, data: ByteArray) {
    val key = keys.firstOrNull { tag.authenticateSectorWithKeyA(sector, it) }
    //val key = keys.firstOrNull { tag.authenticateSectorWithKeyB(sector, it) }
    if (key ≠ null) {
        val blocksCount = tag.getBlockCountInSector(sector)
        val blocksToWrite = data.size / 16
        if (blocksToWrite > blocksCount - 1 || sector == 0) {
            throw IllegalArgumentException("data length should be ≤ ${blocksCount * 16} || zero sector was chosen")
        } else {





        }
    }
}
```

Запись Mifare Classic

```
private fun writeClassic(tag: MifareClassic, keys: Set<ByteArray>, sector: Int, data: ByteArray) {
    val key = keys.firstOrNull { tag.authenticateSectorWithKeyA(sector, it) }
    //val key = keys.firstOrNull { tag.authenticateSectorWithKeyB(sector, it) }
    if (key ≠ null) {
        val blocksCount = tag.getBlockCountInSector(sector)
        val blocksToWrite = data.size / 16
        if (blocksToWrite > blocksCount - 1 || sector == 0) {
            throw IllegalArgumentException("data length should be ≤ ${blocksCount * 16} || zero sector was chosen")
        } else {
            val buffer = ByteArray(16)
            for (block in 0 until blocksToWrite) {
                System.arraycopy(data, 0, buffer, block * 16, 16)
                tag.writeBlock(sector * blocksCount + block, buffer)
            }
            Log.d("Classic", "write finished")
        }
    }
}
```

Чтение и запись Mifare Ultralight

```
private fun readUltralight(tag: MifareUltralight) {
    Log.d("Ultralight", "read Ultralight")
    for (i in 0 until 16 step 4) {
        val data = tag.readPages(i)
        val pageBuffer = ByteArray(4)
        for (page in 0 until 4) {
            System.arraycopy(data, page * 4, pageBuffer, 0, 4)
            Log.d("Ultralight", pageBuffer.toHexString())
        }
    }
}
```

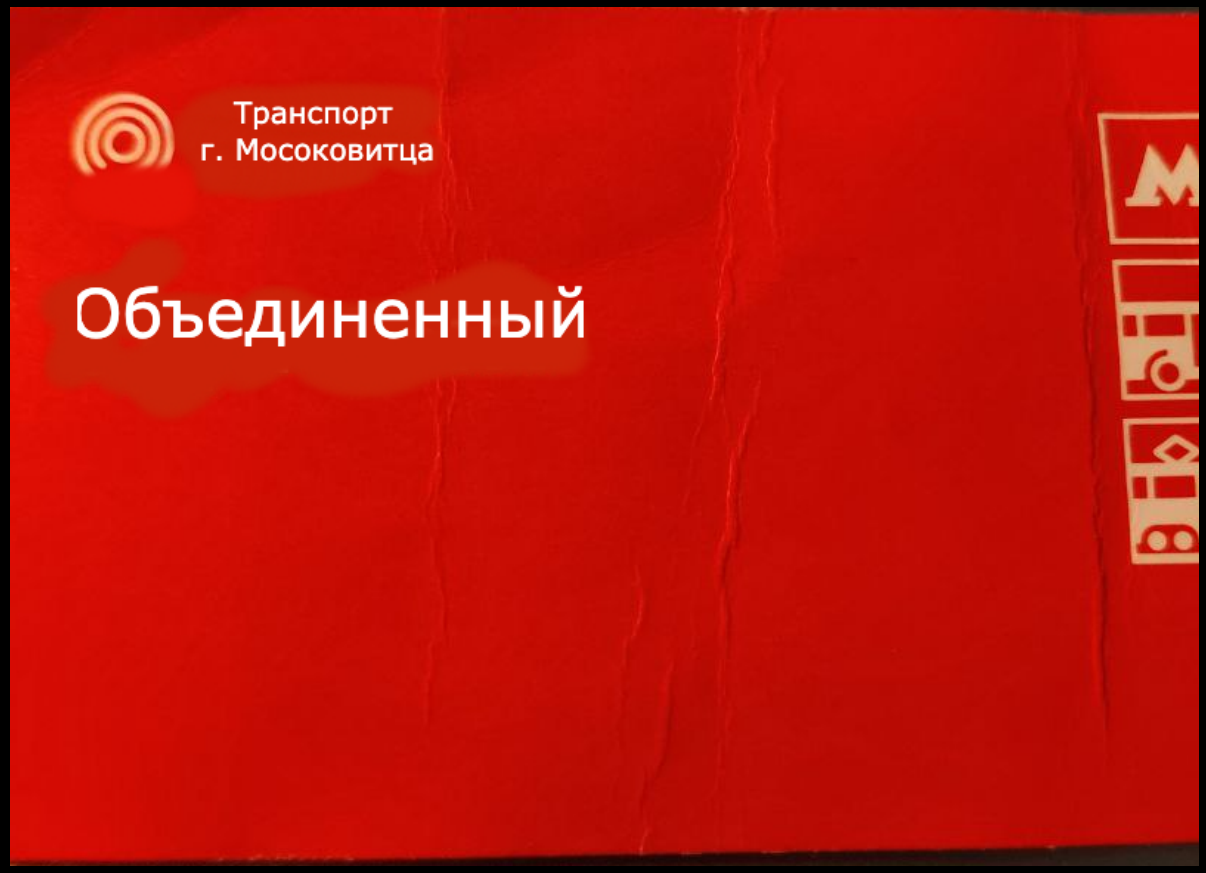
Чтение и запись Mifare Ultralight

```
private fun readUltralight(tag: MifareUltralight) {
    Log.d("Ultralight", "read Ultralight")
    for (i in 0 until 16 step 4) {
        val data = tag.readPages(i)
        val pageBuffer = ByteArray(4)
        for (page in 0 until 4) {
            System.arraycopy(data, page * 4, pageBuffer, 0, 4)
            Log.d("Ultralight", pageBuffer.toHexString())
        }
    }
}

private fun writeUltralight(tag: MifareUltralight, pageOffset: Int, data: ByteArray) {
    tag.writePage(pageOffset, data)
}
```

Что-то пошло не так…

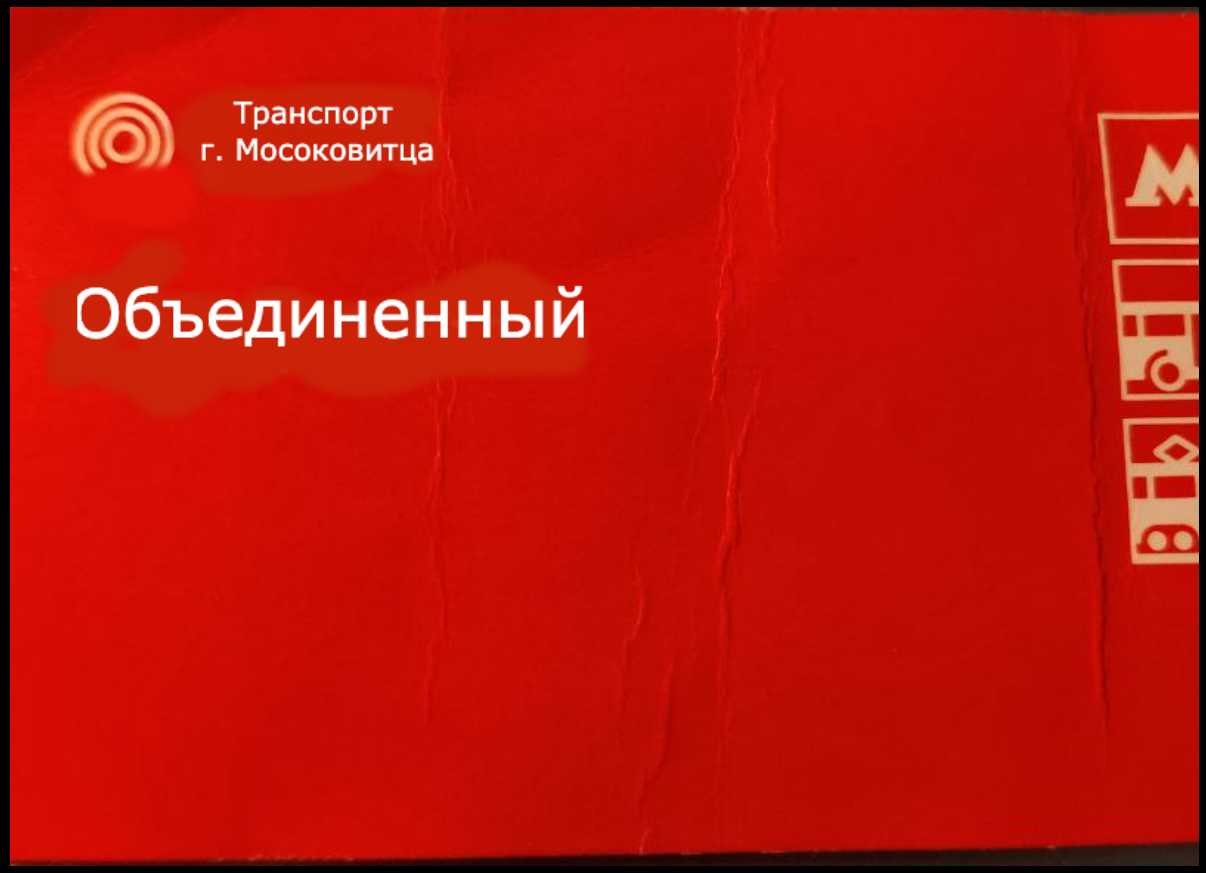
Mifare Ultralight EV1

Mifare Ultralight EV1



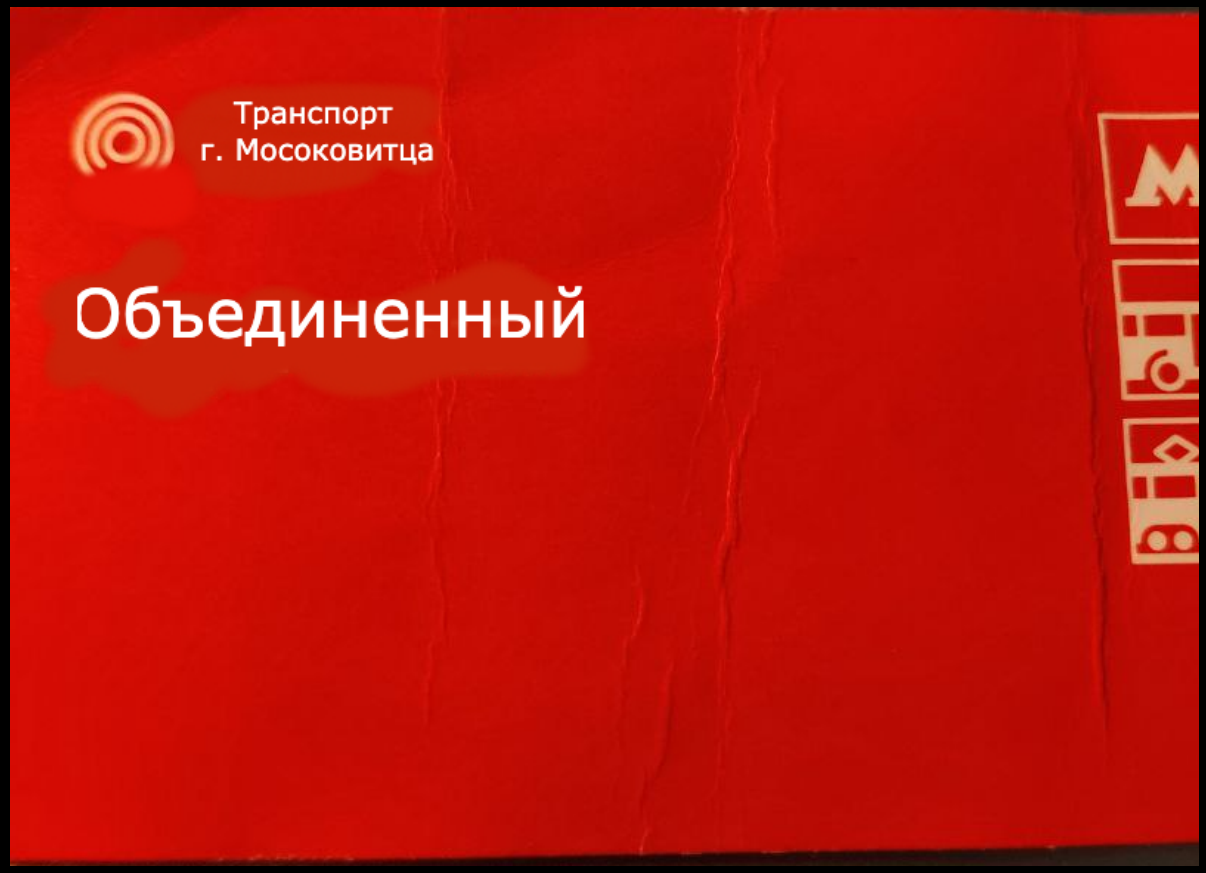
Mifare Ultralight EV1

- ATQA = 0x0044



Mifare Ultralight EV1

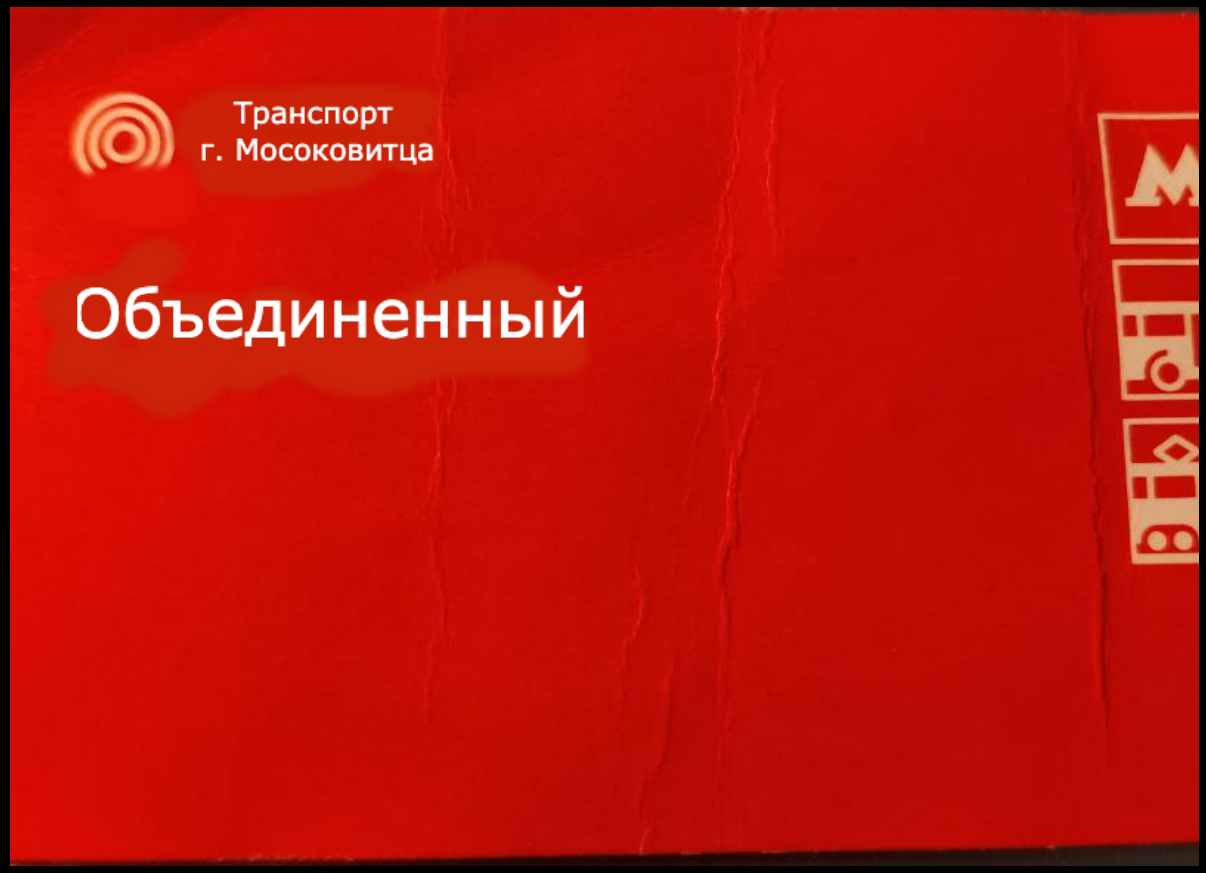
- ATQA = 0x0044
- SAK = 0x0



Mifare Ultralight EV1

- ATQA = 0x0044
- SAK = 0x0

```
private fun handleNfcA(nfcA: NfcA) {
    if (nfcA.atqa.toHexString() == "4400"
        && nfcA.sak == 0.toShort()) {

    }
}
```

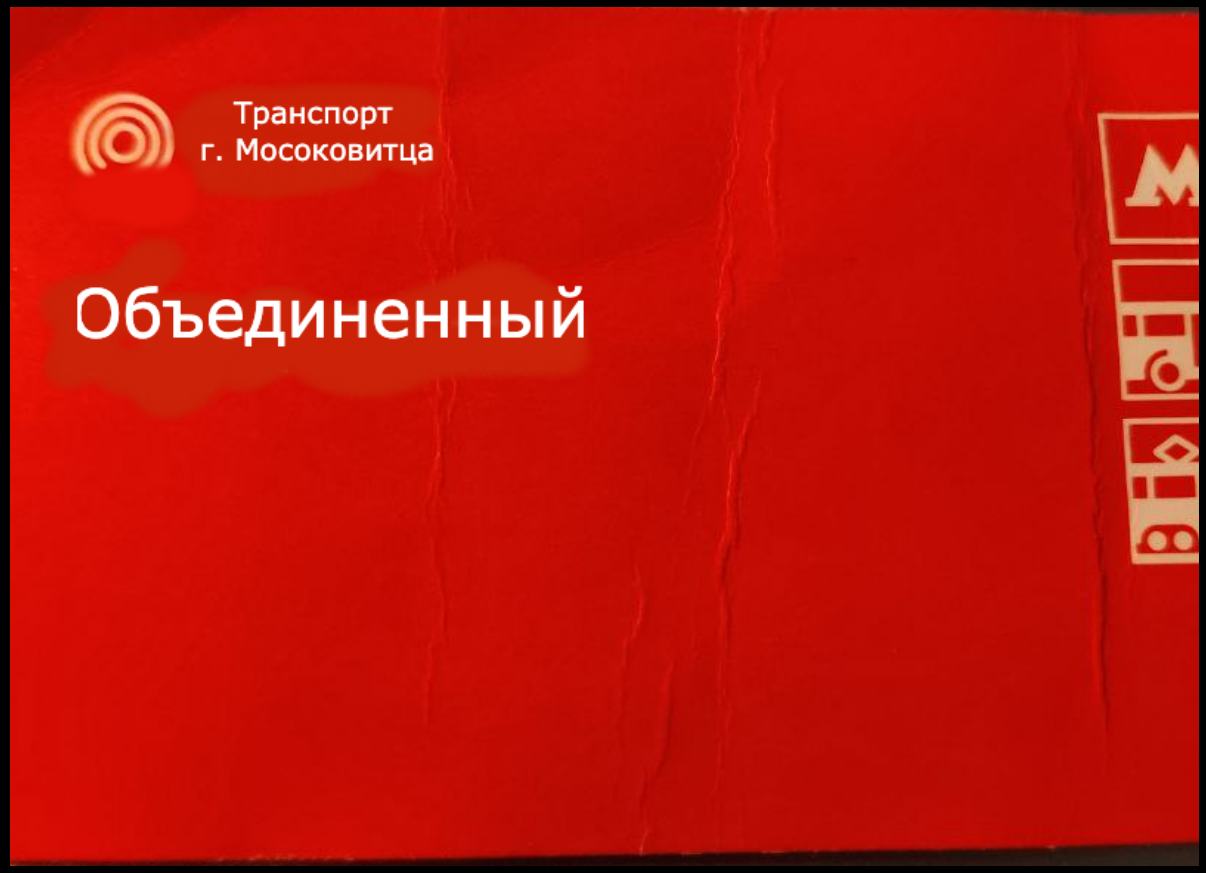


Mifare Ultralight EV1

- ATQA = 0x0044
- SAK = 0x0

```
private fun handleNfcA(nfcA: NfcA) {
    if (nfcA.atqa.toHexString() == "4400"
        && nfcA.sak == 0.toShort()) {

    }
}
```

```
public byte[] readPages(int pageOffset) throws IOException {
    validatePageIndex(pageOffset);
    checkConnected();

    byte[] cmd = { 0x30, (byte) pageOffset};
    return transceive(cmd, false);
}
```



Mifare Ultralight EV1

```
// MifareUltralight read 4 Page
private fun NfcA.ultralightReadCmd(pageOffset: Int) : ByteArray {
    val cmd = byteArrayOf(0x30, pageOffset.toByte())
    return transceive(cmd)
}

// MifareUltralight write one page
private fun NfcA.writePage(page: Int, data: ByteArray) {





}
```

Mifare Ultralight EV1

```
// MifareUltralight read 4 Page
private fun NfcA.ultralightReadCmd(pageOffset: Int) : ByteArray {
    val cmd = byteArrayOf(0x30, pageOffset.toByte())
    return transceive(cmd)
}

// MifareUltralight write one page
private fun NfcA.writePage(page: Int, data: ByteArray) {
    val cmd = ByteArray(data.size + 2)
    cmd[0] = 0xA2.toByte()
    cmd[1] = page.toByte()
    System.arraycopy(data, 0, cmd, 2, data.size)

    transceive(cmd)
}
```

Эмуляция карт

Эмуляция карт

```
class MyHostApduService : HostApduService() {
    override fun processCommandApdu(commandApdu: ByteArray, extras: Bundle?): ByteArray {
        ...
    }
    override fun onDeactivated(reason: Int) {
        ...
    }
}
```

Транспортные карты

Транспортные карты

Транспортные карты

- Ответственность ст. 327 УК

Транспортные карты

- Ответственность ст. 327 УК
- Мосоковица (Опять двойка)

Транспортные карты

- Ответственность ст. 327 УК
- Мосоковица (Опять двойка)
- Сыр-Пицценбург (Одуванчик)

Опять двойка

Опять двойка

- Mifare Plus S - SL1 (Classic 2k)

Опять двойка

- Mifare Plus S - SL1 (Classic 2k)
- 1 сектор - транспортное приложение подземки (банковские карты)

Опять двойка

- Mifare Plus S - SL1 (Classic 2k)
- 1 сектор - транспортное приложение подземки (банковские карты)
- 2 сектор - электрички (бланк)

Опять двойка

- Mifare Plus S - SL1 (Classic 2k)
- 1 сектор - транспортное приложение подземки (банковские карты)
- 2 сектор - электрички (бланк)
- 3 сектор - электрички (тикет)

Опять двойка

- Mifare Plus S - SL1 (Classic 2k)
- 1 сектор - транспортное приложение подземки (банковские карты)
- 2 сектор - электрички (бланк)
- 3 сектор - электрички (тикет)
- 4 сектор - наземка

Опять двойка

- Mifare Plus S - SL1 (Classic 2k)
- 1 сектор - транспортное приложение подземки (банковские карты)
- 2 сектор - электрички (бланк)
- 3 сектор - электрички (тикет)
- 4 сектор - наземка
- 7 сектор - объединенный тикет

Опять двойка

- Mifare Plus S - SL1 (Classic 2k)
- 1 сектор - транспортное приложение подземки (банковские карты)
- 2 сектор - электрички (бланк)
- 3 сектор - электрички (тикет)
- 4 сектор - наземка
- 7 сектор - объединенный тикет
- 8 сектор - кошелек

Опять двойка

- Mifare Plus S - SL1 (Classic 2k)
- 1 сектор - транспортное приложение подземки (банковские карты)
- 2 сектор - электрички (бланк)
- 3 сектор - электрички (тикет)
- 4 сектор - наземка
- 7 сектор - объединенный тикет
- 8 сектор - кошелек
- 9 сектор - централизованные диаметральные кольца

Опять двойка

- Mifare Plus S - SL1 (Classic 2k)
- 1 сектор - транспортное приложение подземки (банковские карты)
- 2 сектор - электрички (бланк)
- 3 сектор - электрички (тикет)
- 4 сектор - наземка
- 7 сектор - объединенный тикет
- 8 сектор - кошелек
- 9 сектор - централизованные диаметральные кольца
- Приложения: ТКМ

Опять двойка - 8 сектор

Опять двойка - 8 сектор

- ID считывателя

Опять двойка - 8 сектор

- ID считывателя
- Тип тикета

Опять двойка - 8 сектор

- ID считывателя
- Тип тикета
- Номер карты

Опять двойка - 8 сектор

- ID считывателя
- Тип тикета
- Номер карты
- Дата и время активации

Опять двойка - 8 сектор

- ID считывателя
- Тип тикета
- Номер карты
- Дата и время активации
- Дата и время пересадки

Опять двойка - 8 сектор

- ID считывателя
- Тип тикета
- Номер карты
- Дата и время активации
- Дата и время пересадки
- Тариф

Опять двойка - 8 сектор

- ID считывателя
- Тип тикета
- Номер карты
- Дата и время активации
- Дата и время пересадки
- Тариф
- Баланс

Опять двойка - 8 сектор

- ID считывателя
- Тип тикета
- Номер карты
- Дата и время активации
- Дата и время пересадки
- Тариф
- Баланс
- Бит блокировки

Опять двойка - 8 сектор

- ID считывателя
- Тип тикета
- Номер карты
- Дата и время активации
- Дата и время пересадки
- Тариф
- Баланс
- Бит блокировки
- Имитовставка

Данные

Данные

```
Sector: 8

45DB1016949     96FEAA40000000000
010F6D6BBE09E400B4462000DE5C5D4F
010F6D6BBE09E400B4462000DE5C5D4F
```

Данные

- Номер - 16949XXXX

```
Sector: 8

45DB1016949      96FEAA40000000000
010F6D6BBE09E400B4462000DE5C5D4F
010F6D6BBE09E400B4462000DE5C5D4F
```

Данные

```
Sector: 8

45DB1016949      96FEAA40000000000
010F6D6BBE09E400B4462000DE5C5D4F
010F6D6BBE09E400B4462000DE5C5D4F
```

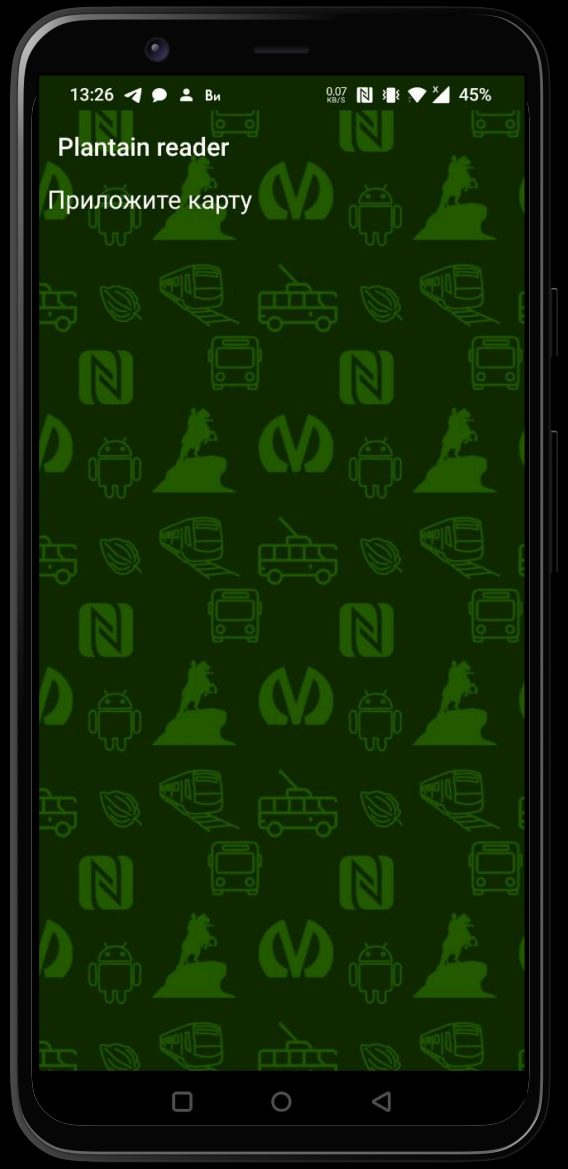
- Номер - 16949XXXX
- Баланс - hexToDec(B446 / 19) = 1846

Данные

```
Sector: 8

45DB1016949      96FEAA40000000000
010F6D6BBE09E400B4462000DE5C5D4F
010F6D6BBE09E400B4462000DE5C5D4F
```

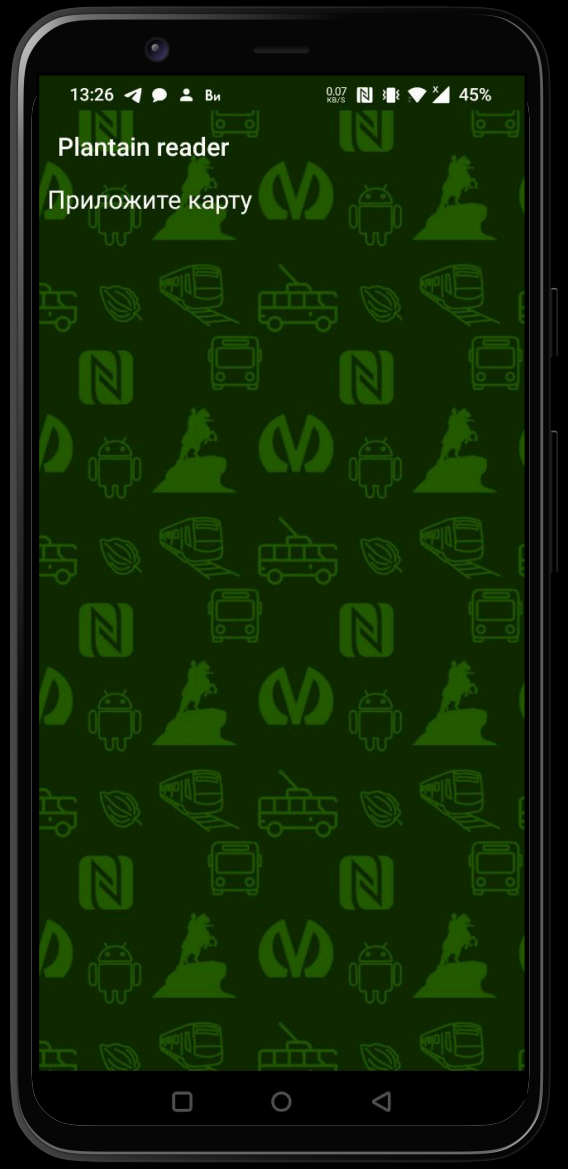
- Номер - 16949XXXX
- Баланс - hexToDec(B446 / 19) = 1846
- ID считывателя

Одуванчик



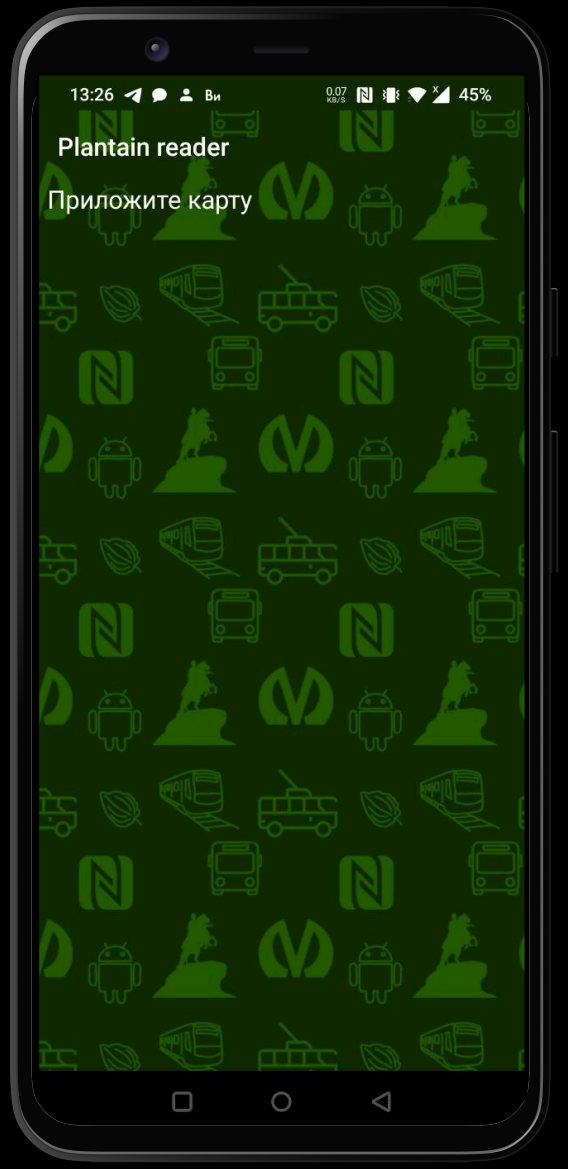
Одуванчик

- Mifare Plus 4k



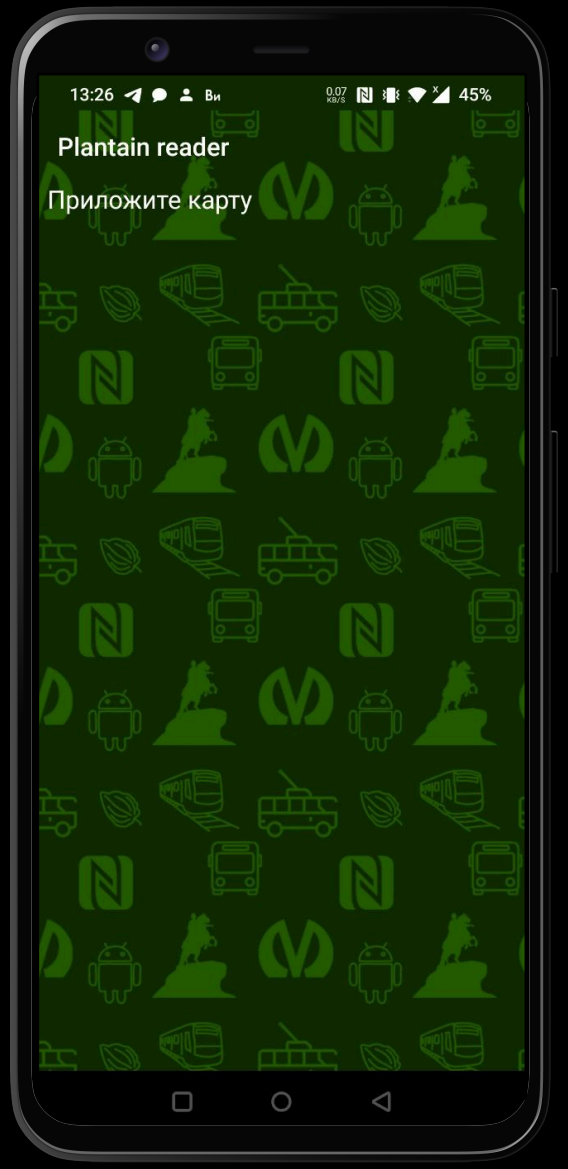
Одуванчик

- Mifare Plus 4k
- 4 и 5 сектор



Одуванчик

- Mifare Plus 4k
- 4 и 5 сектор
- Plantain reader



Уязвимости

Уязвимости

Уязвимости

- Атака повторного копирования

Уязвимости

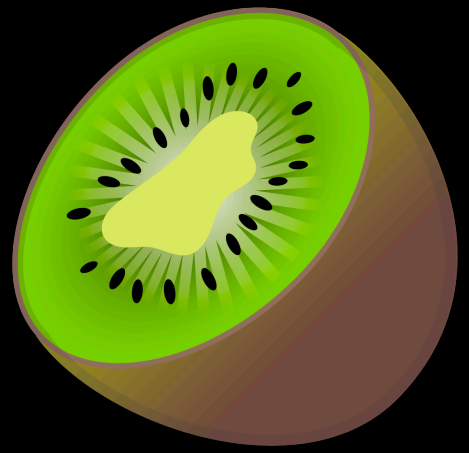
- Атака повторного копирования
- Перенос на другой носитель (Одуванчик)

Уязвимости

- Атака повторного копирования
- Перенос на другой носитель (Одуванчик)
- Пополнение на 1 тугрик (Опять двойка)

Уязвимости

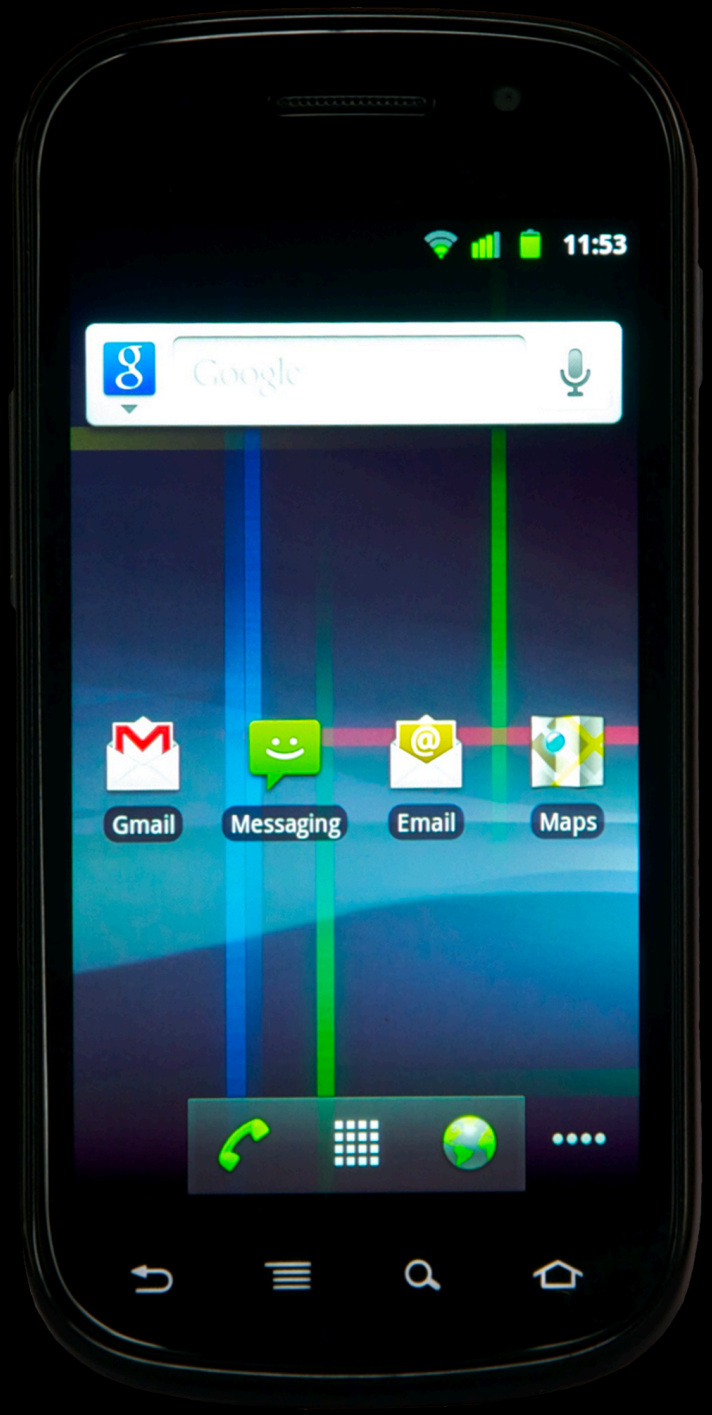
- Атака повторного копирования
- Перенос на другой носитель (Одуванчик)
- Пополнение на 1 тугрик (Опять двойка)



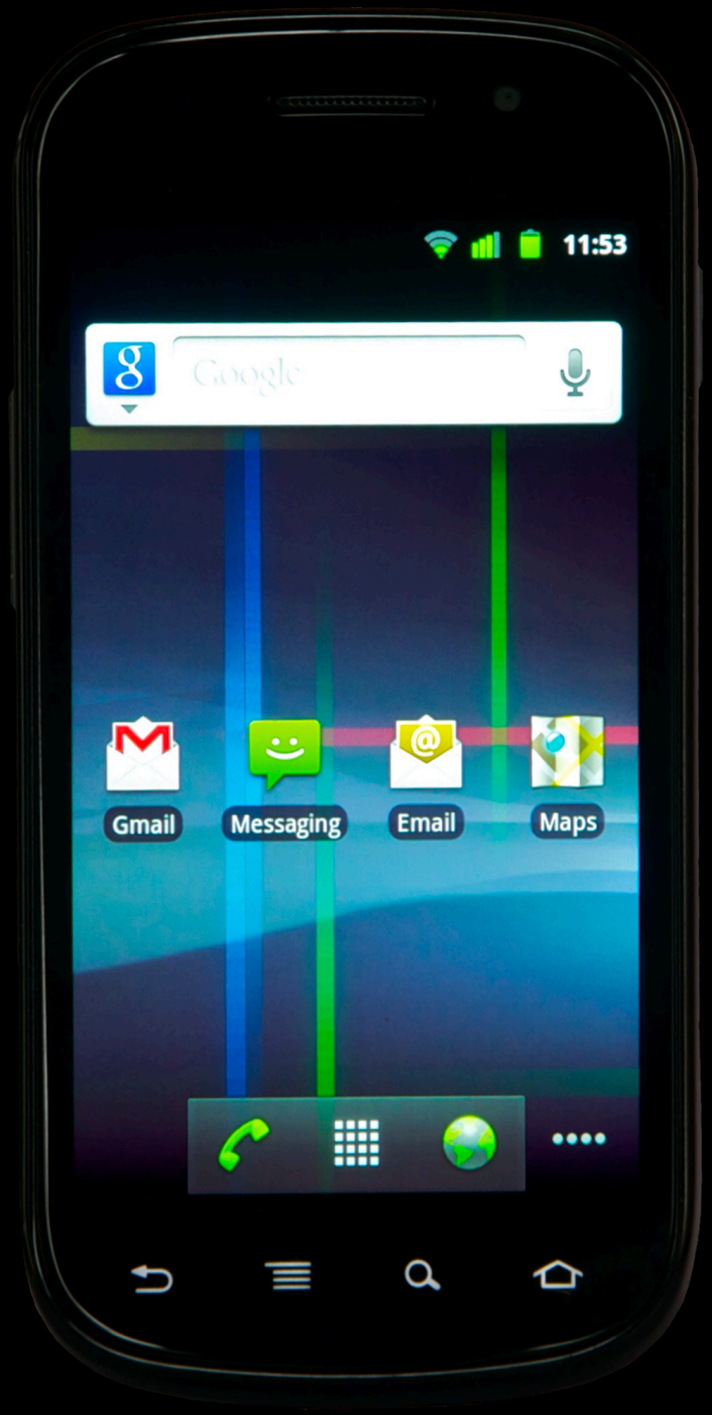
Китайская магия

Mifare Classic Zero

- Изменяемый UID
- UID 4 и 7 байт
- Игнор битов доступа*

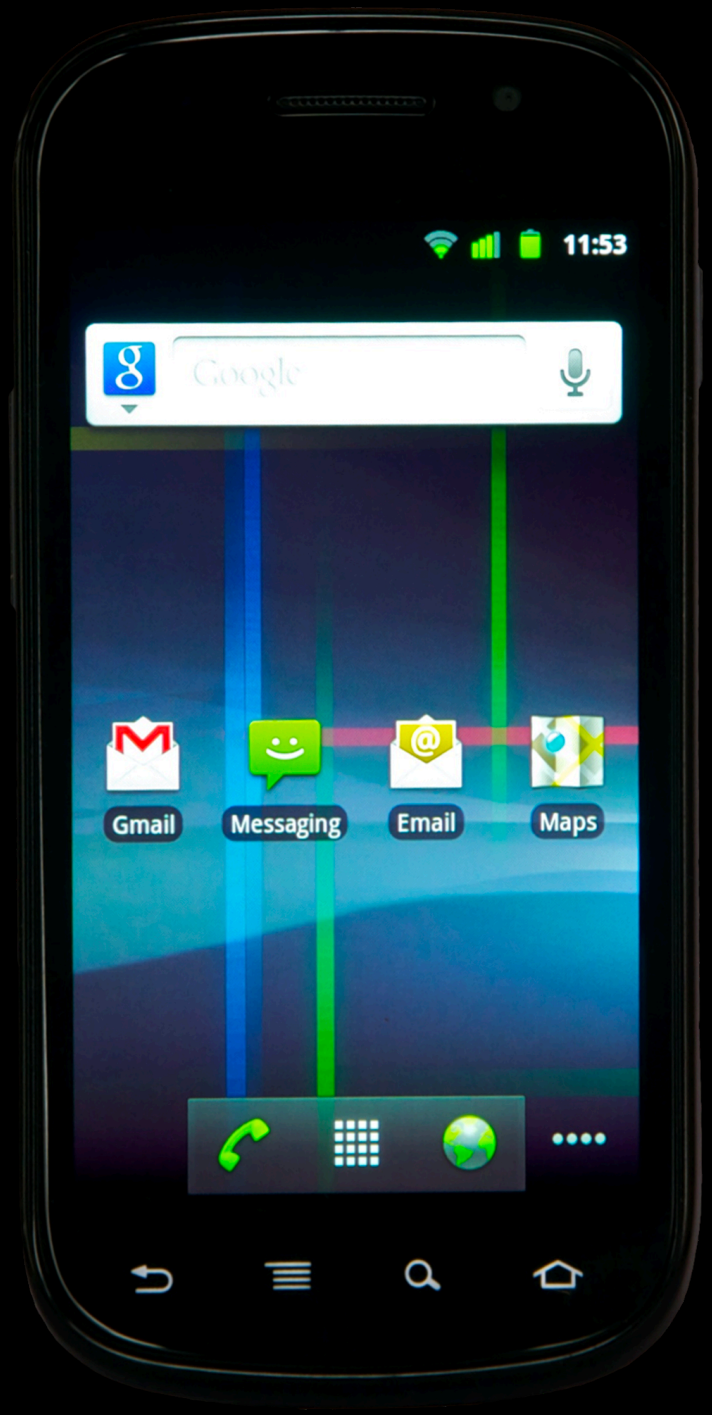


Как его зовут?



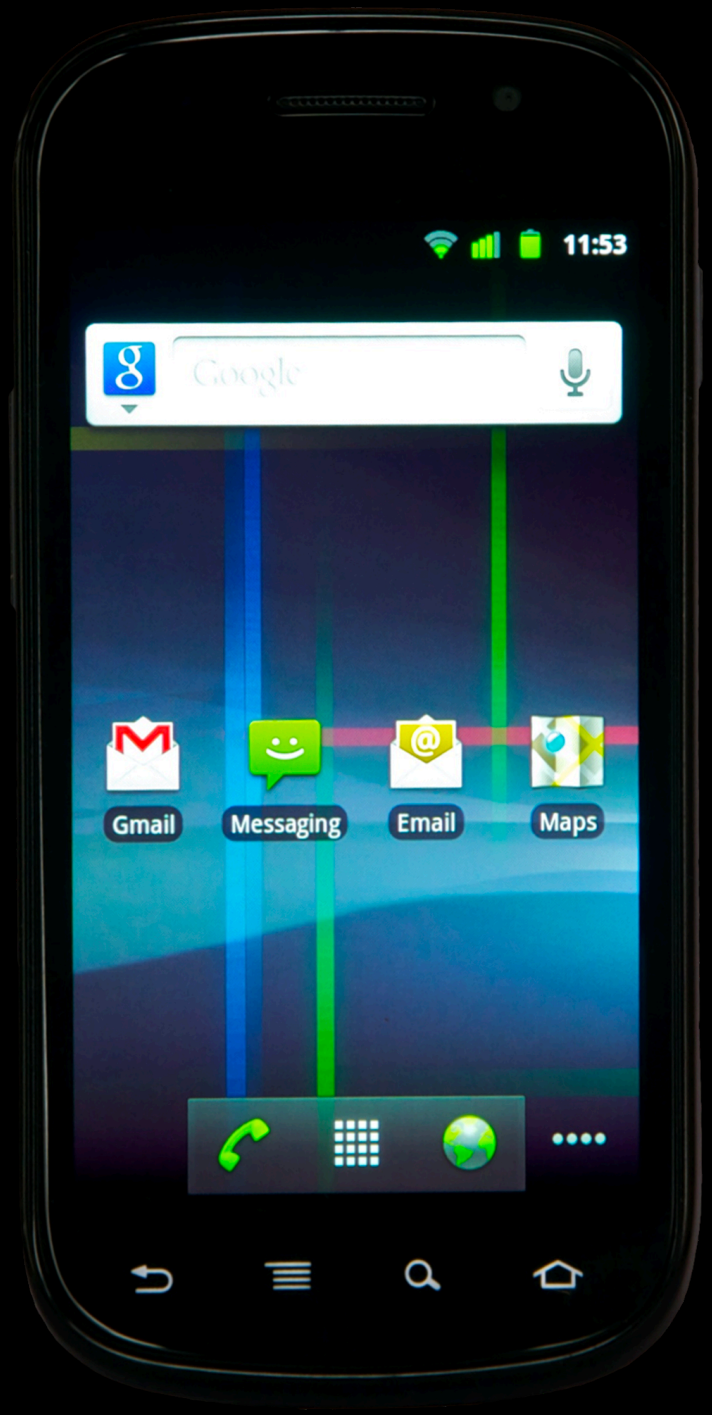
Как его зовут?

- Nexus S



Как его зовут?

- Nexus S
- Работает с картами с UID 0x8X000000



Как его зовут?

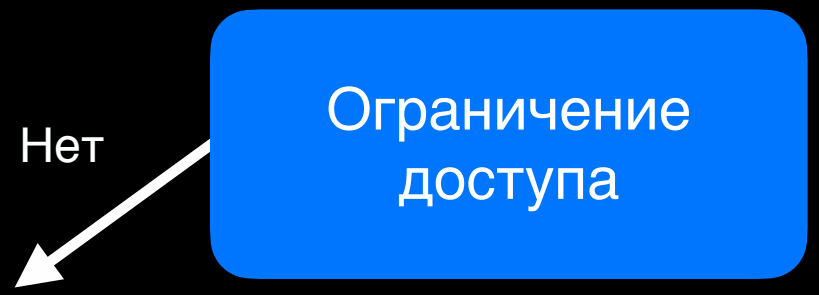
- Nexus S
- Работает с картами с UID 0x8X000000
- Android Kitkat 4.4

Какие карты выбрать?

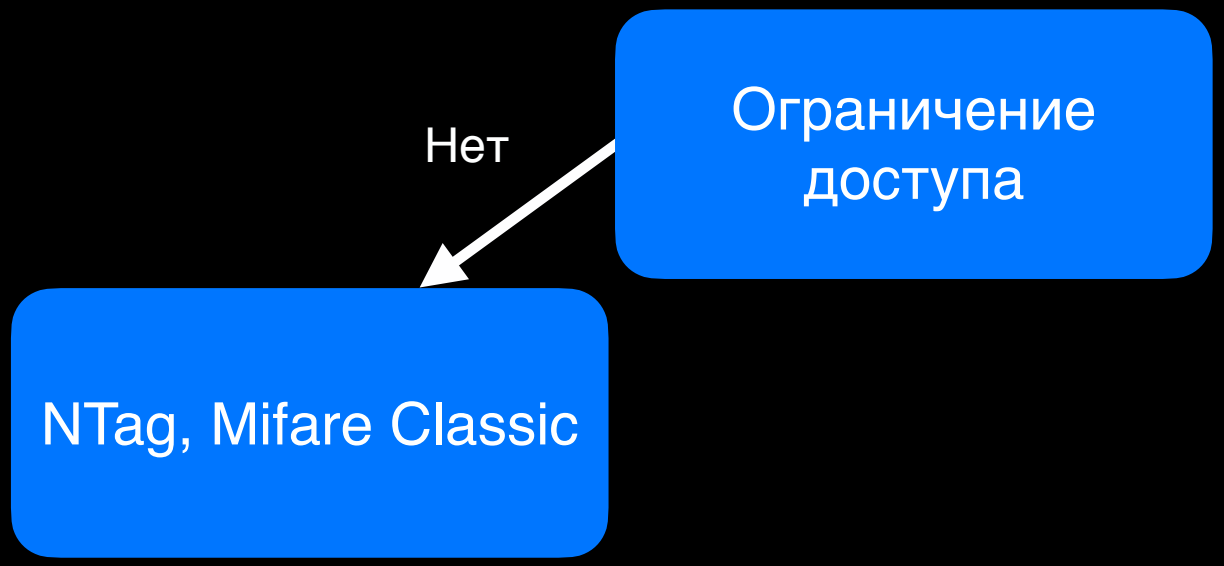
Какие карты выбрать?

Ограничение доступа

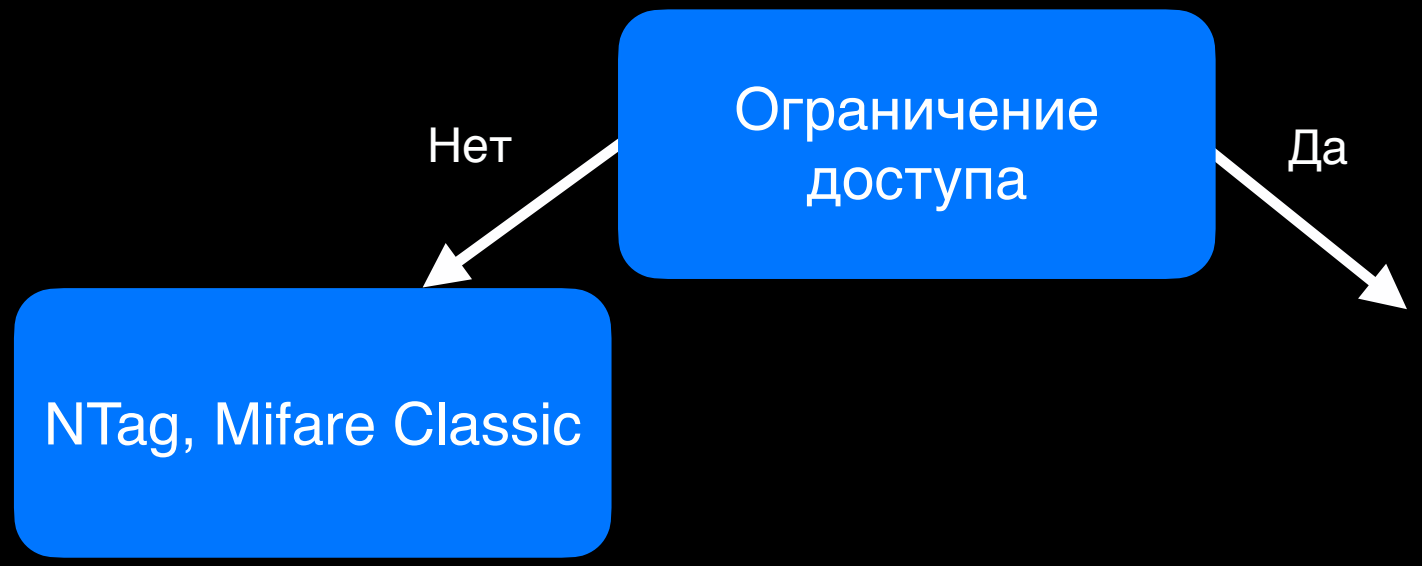
Какие карты выбрать?



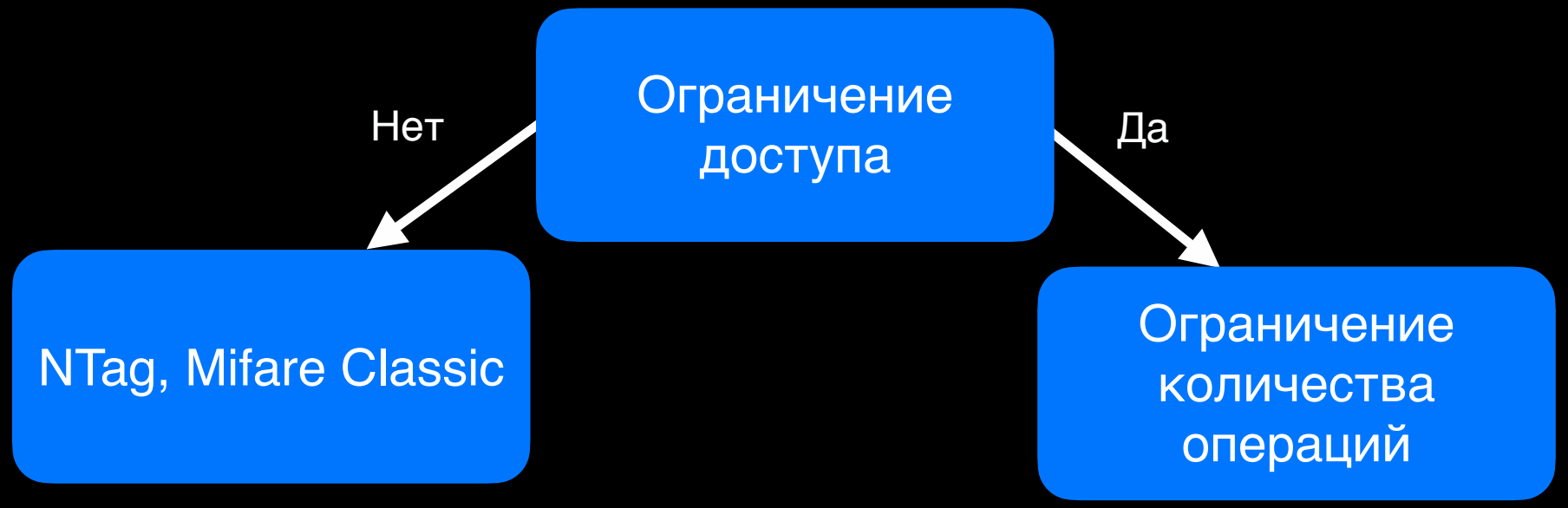
Какие карты выбрать?



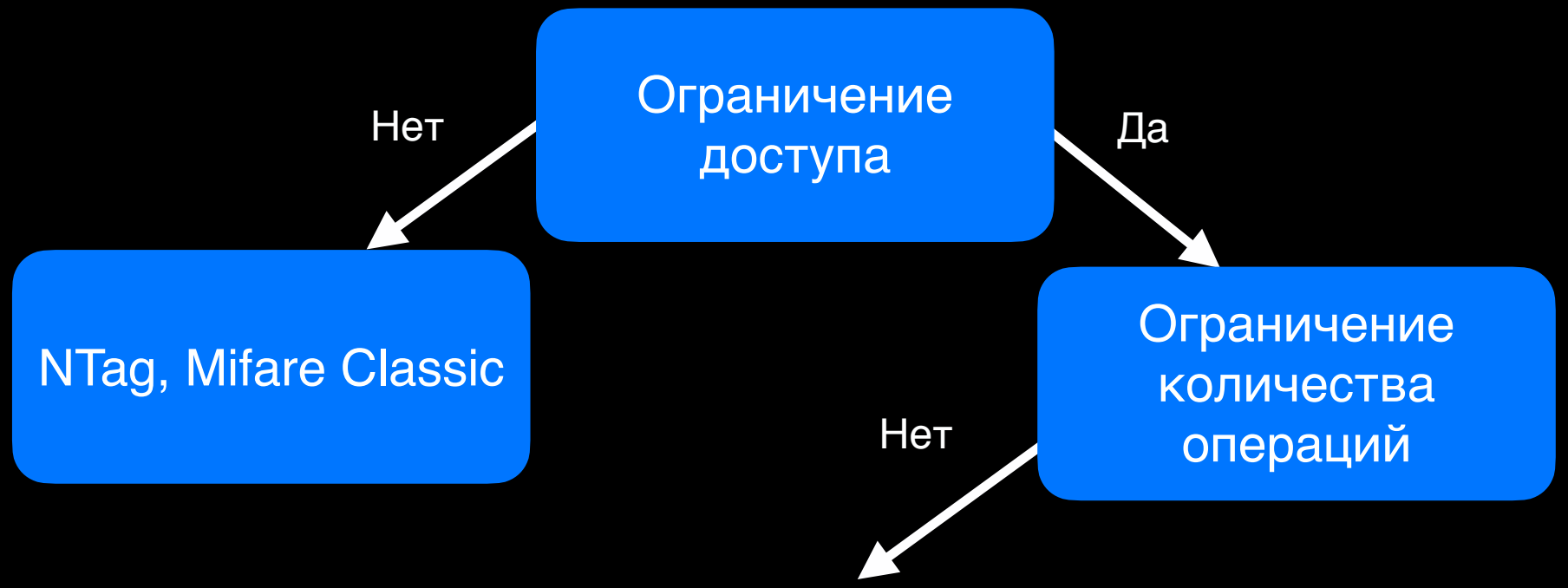
Какие карты выбрать?



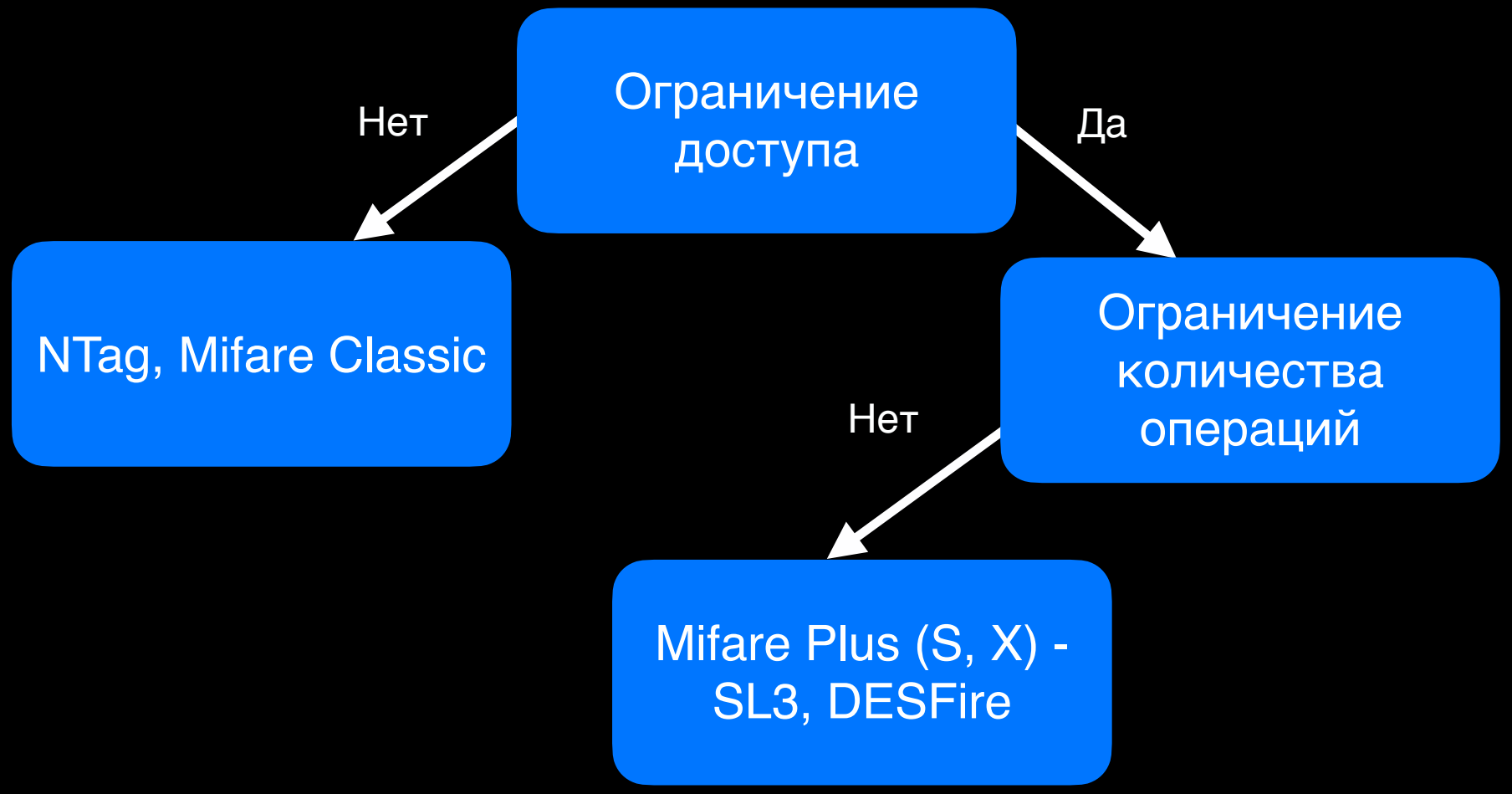
Какие карты выбрать?



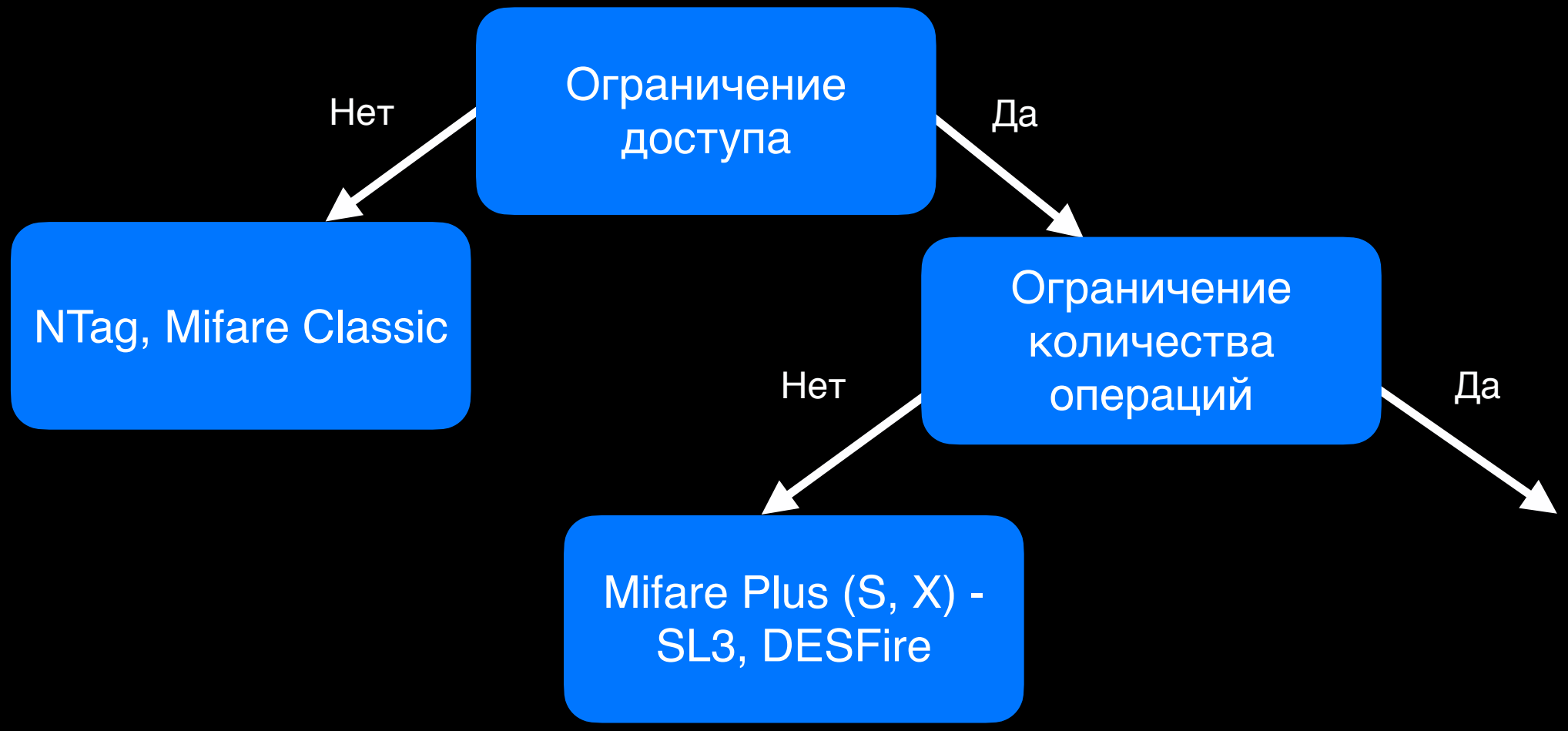
Какие карты выбрать?



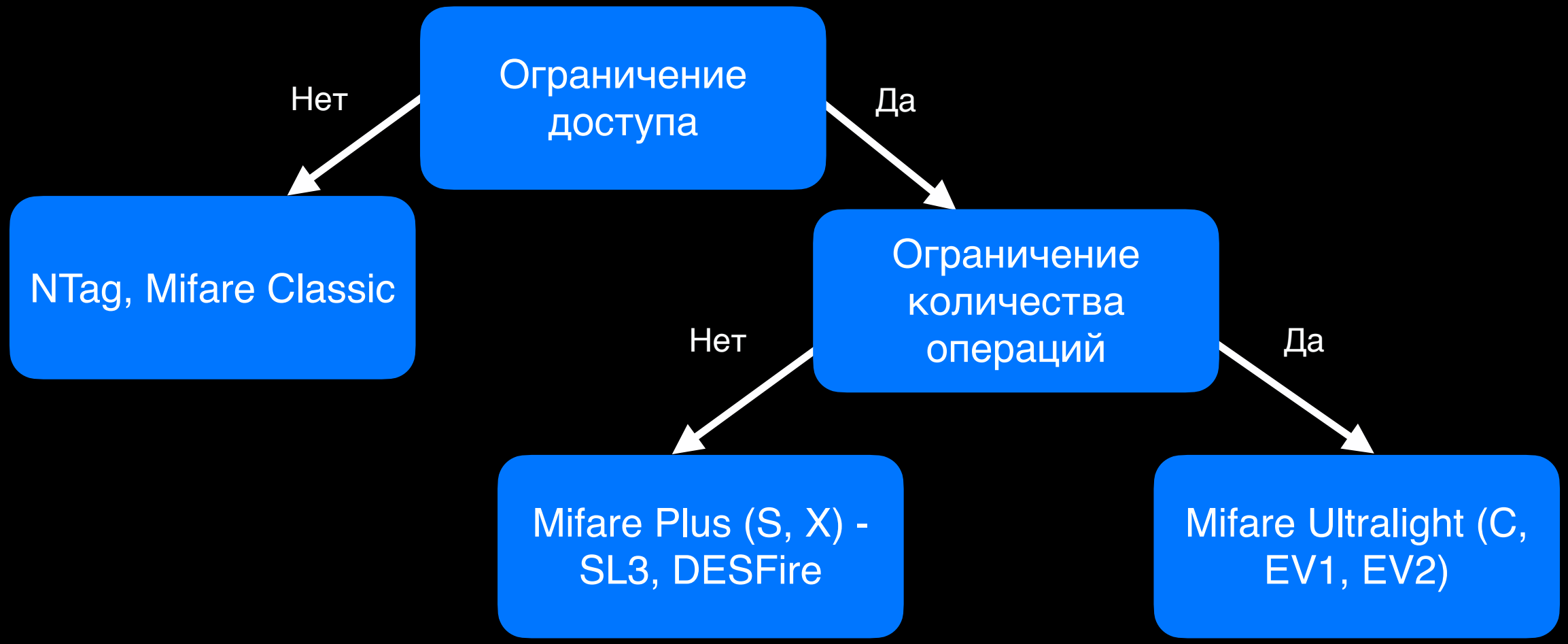
Какие карты выбрать?



Какие карты выбрать?



Какие карты выбрать?



iOS Core NFC

iOS Core NFC

- iOS 11

iOS Core NFC

- iOS 11
- iOS13+

iOS Core NFC

- iOS 11
- iOS13+
- Mifare, Felica, ISO15693, ISO7816

iOS Core NFC

- iOS 11
- iOS13+
- Mifare, Felica, ISO15693, ISO7816
- Эмуляция карт недоступна

iOS Core NFC

- iOS 11
- iOS13+
- Mifare, Felica, ISO15693, ISO7816
- Эмуляция карт недоступна
- Примеры от NXP

Выводы

1. Хорошая документация
2. 13,56 МГц
3. SDK от производителей меток
4. APDU - команды
5. ATQA && SAK
6. Android > iOS

https://vk.cc/ciGnDS

v.kozhushko@corp.vk.com

Беспроводная зарядка

1. LED charging indicator light
2. Non-slip pad surface
3. 7.5W transmitter coil
4. Fanless design for quiet operation
5. Wireless charging chipset controls the flow of electricity
6. Thermal protection sensor can dial back power for safer operation
7. Foreign object detection circuit to prevent conductive materials from receiving power from the charger