Шифруем файлы

Андрей Матюшин / VK Tech

Андрей Матюшин
тг - @Hey_O

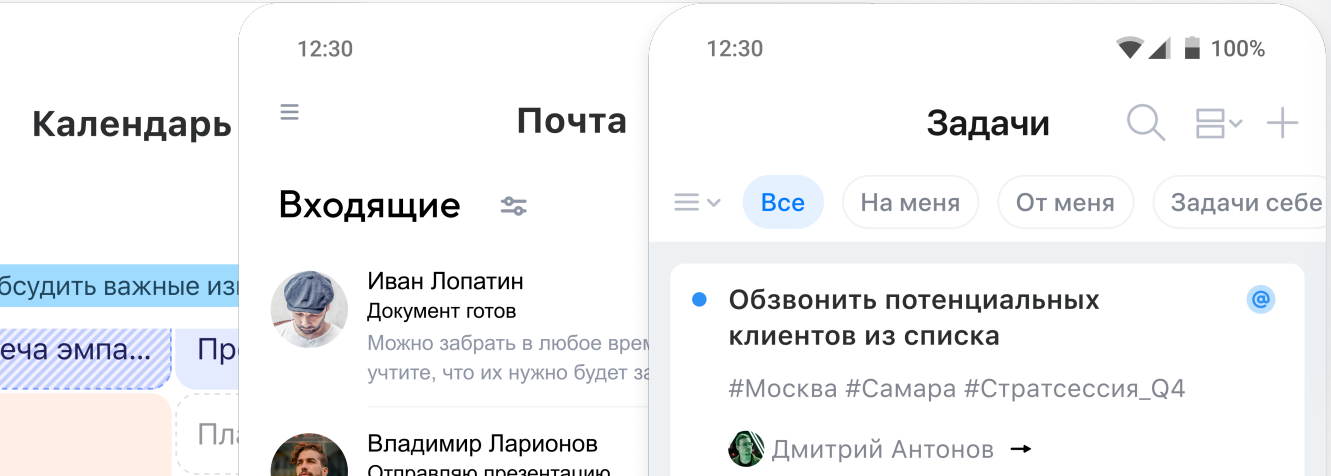
В разработке с 2017 года

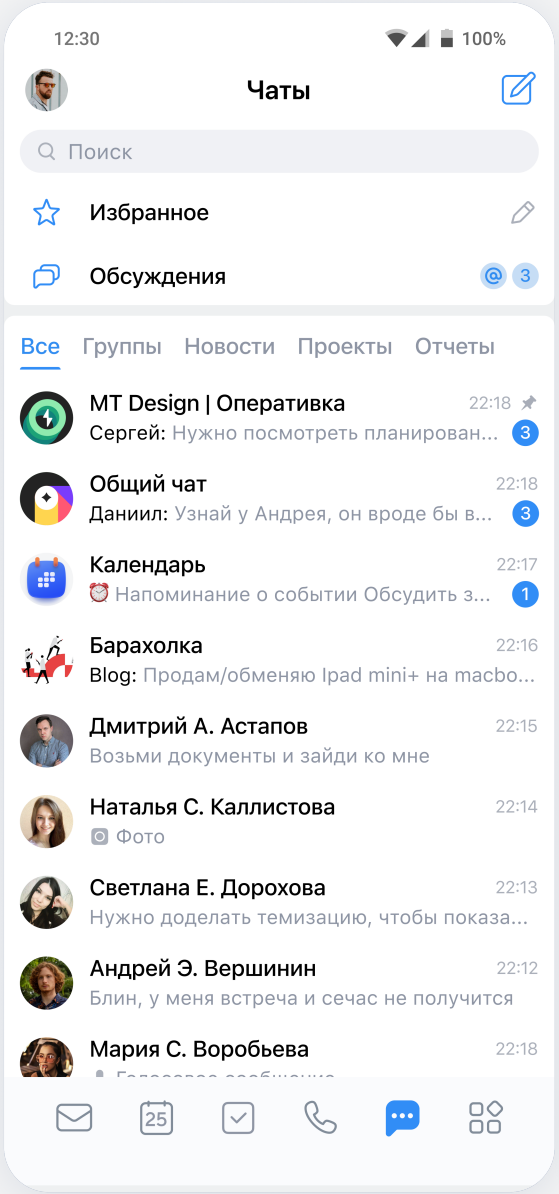
Последние 4.5 года работаю в VK

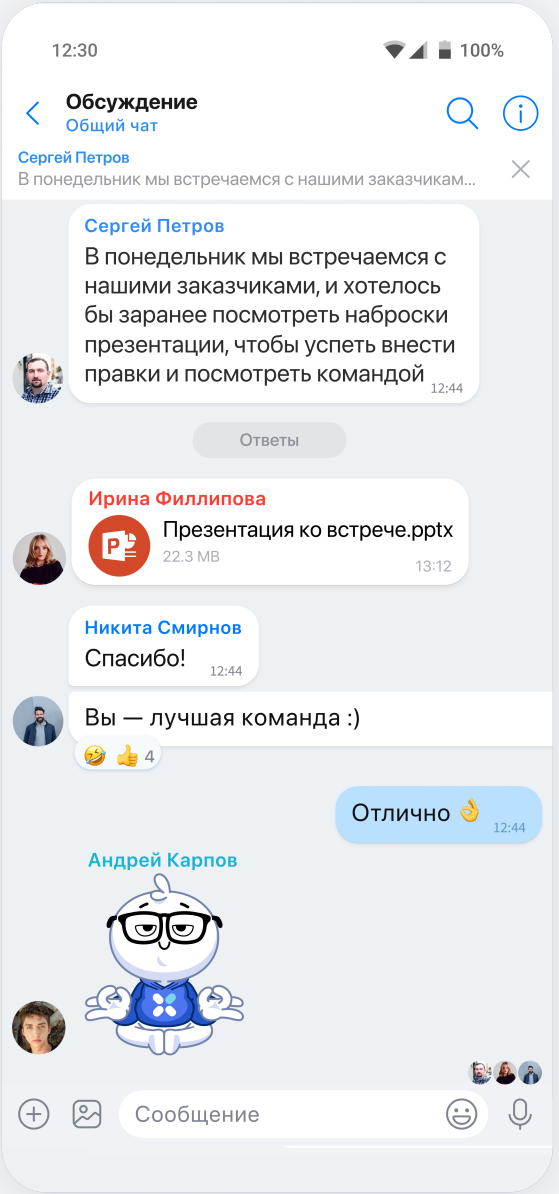
Разрабатываю B2B приложение для коммуникаций

О проекте

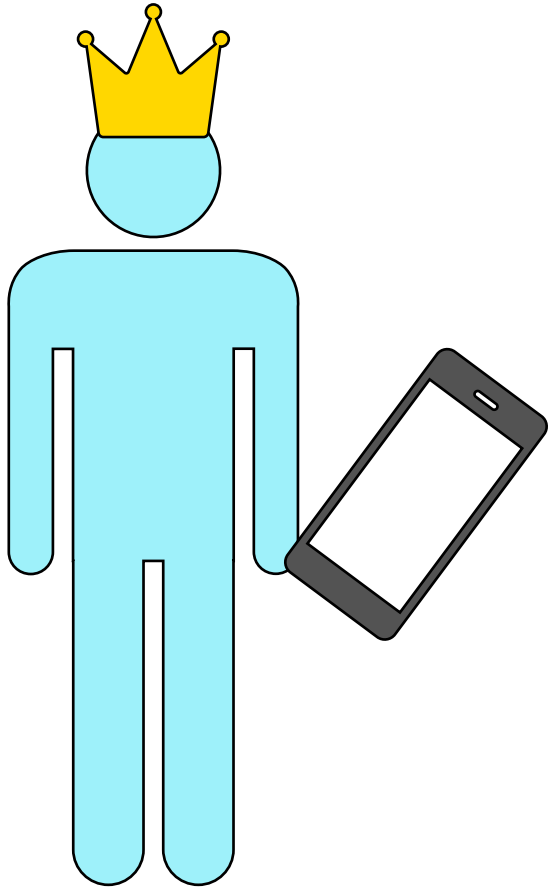
VK Teams



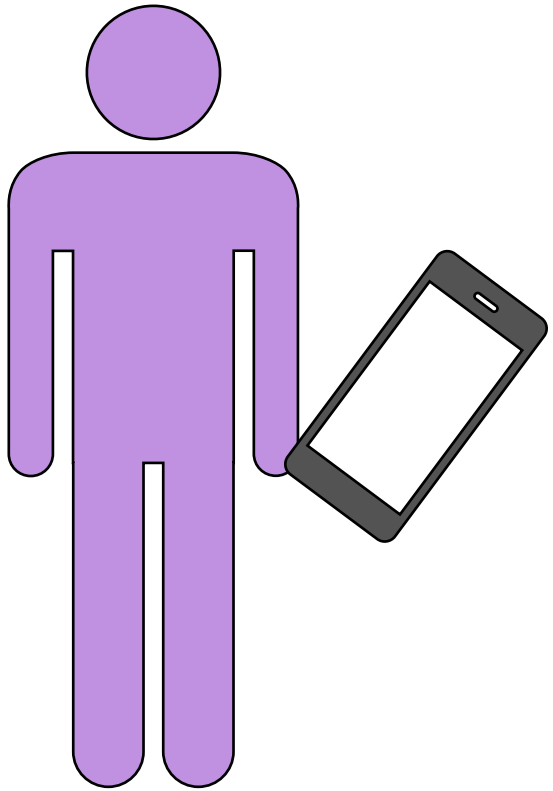




Зачем шифровать файлы локально?



Зачем шифровать файлы локально?



Локальное шифрование - один элемент общей безопасности системы

А оно вам надо?

Full-Disk Encryption

- Был актуален с API 19 и по API 28 включительно

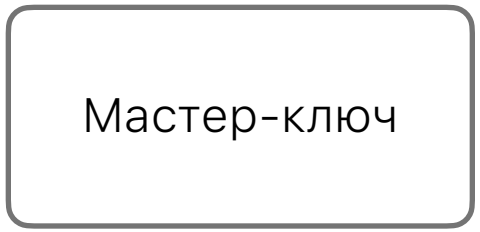
Full-Disk Encryption

- Был актуален с API 19 и по API 28 включительно
- Сразу все данные расшифровываются при загрузке устройства

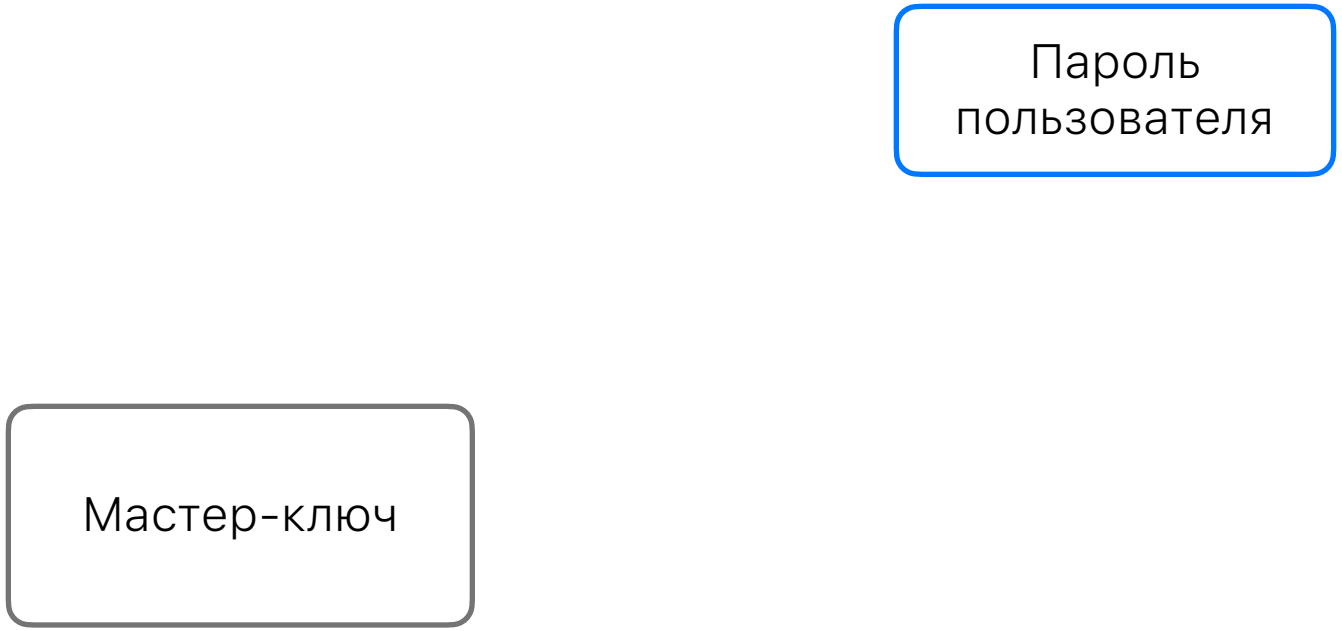
Full-Disk Encryption

- Был актуален с API 19 и по API 28 включительно
- Сразу все данные расшифровываются при загрузке устройства
- После перезагрузки устройства и до разблокировки не могли работать важные сервисы

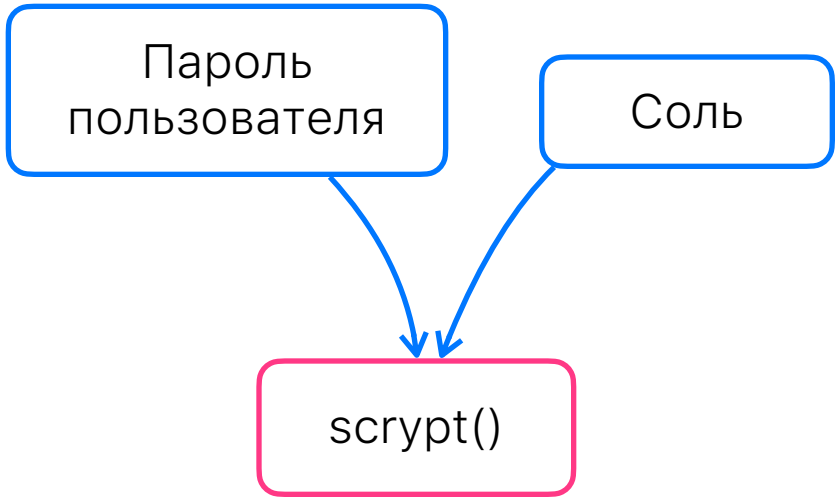
Full-Disk Encryption



Full-Disk Encryption

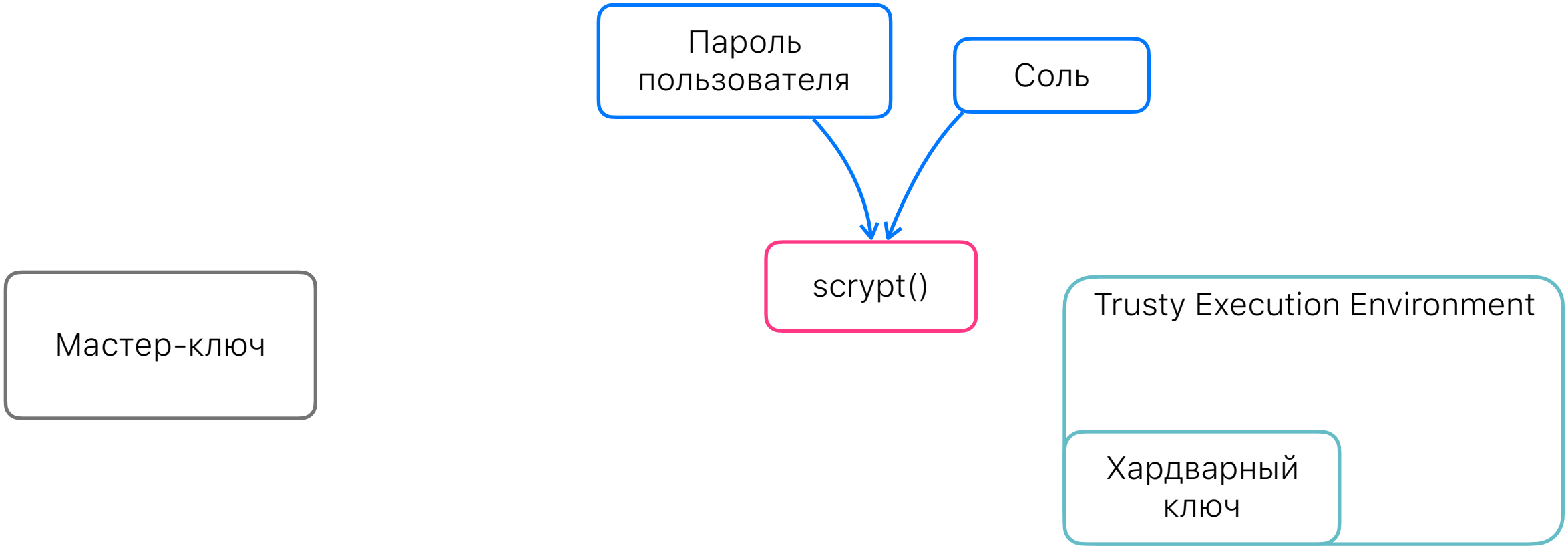


Full-Disk Encryption

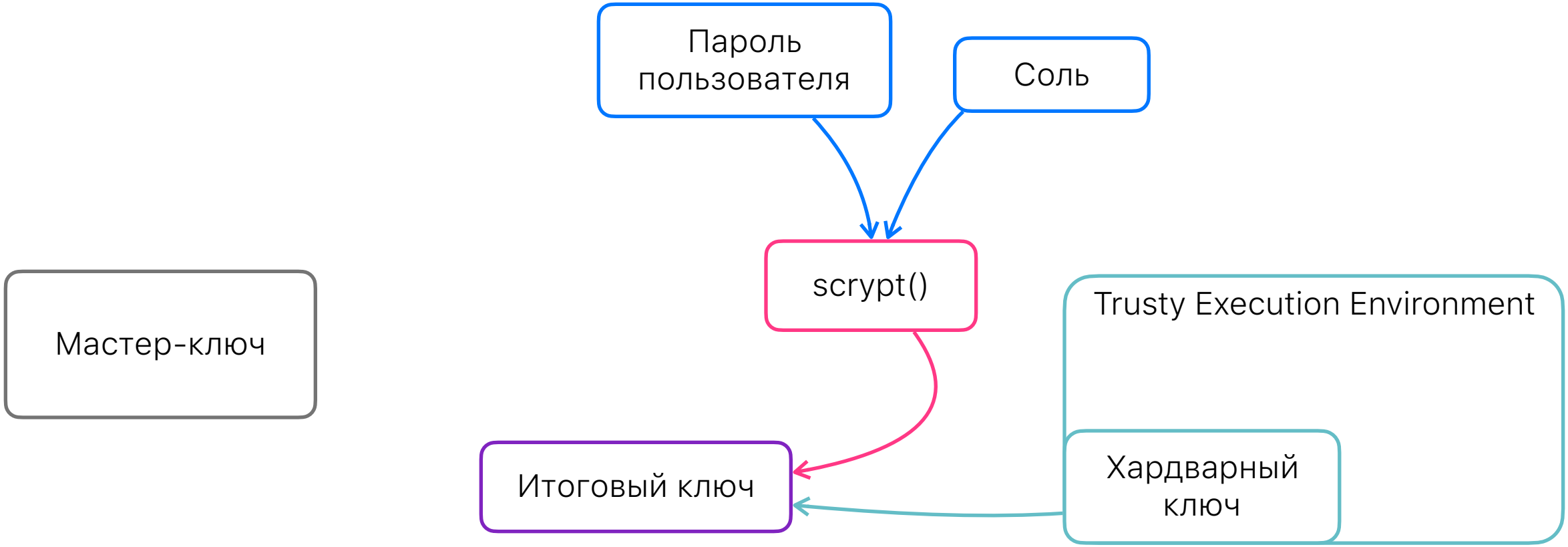


Мастер-ключ

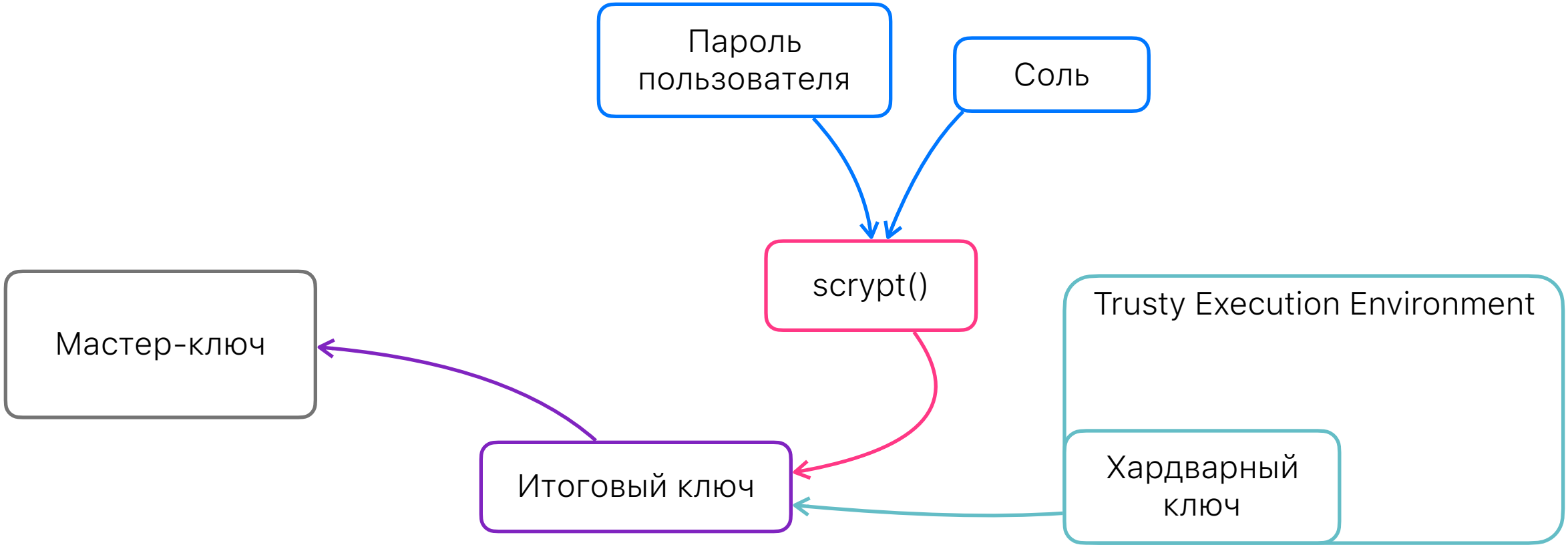
Full-Disk Encryption



Full-Disk Encryption



Full-Disk Encryption



Full-Disk Encryption

```
274
275   #define DEFAULT_PASSWORD "default_password"
276
```

File-Based Encryption

- Появился в API 24. Стал обязательным с API 29

File-Based Encryption

- Появился в API 24. Стал обязательным с API 29
- Direct Boot

File-Based Encryption

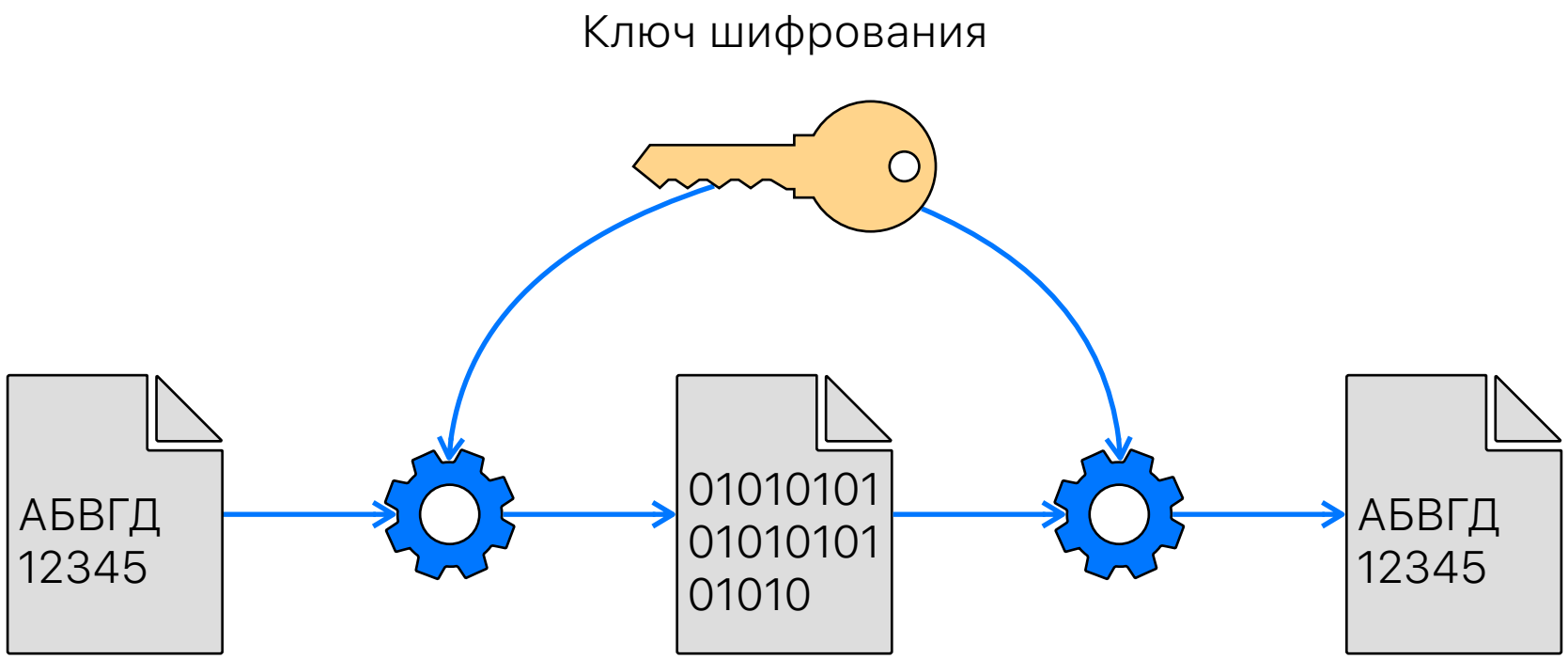
- Появился в API 24. Стал обязательным с API 29
- Direct Boot
- Разные зоны на диске шифруются разными ключами

File-Based Encryption

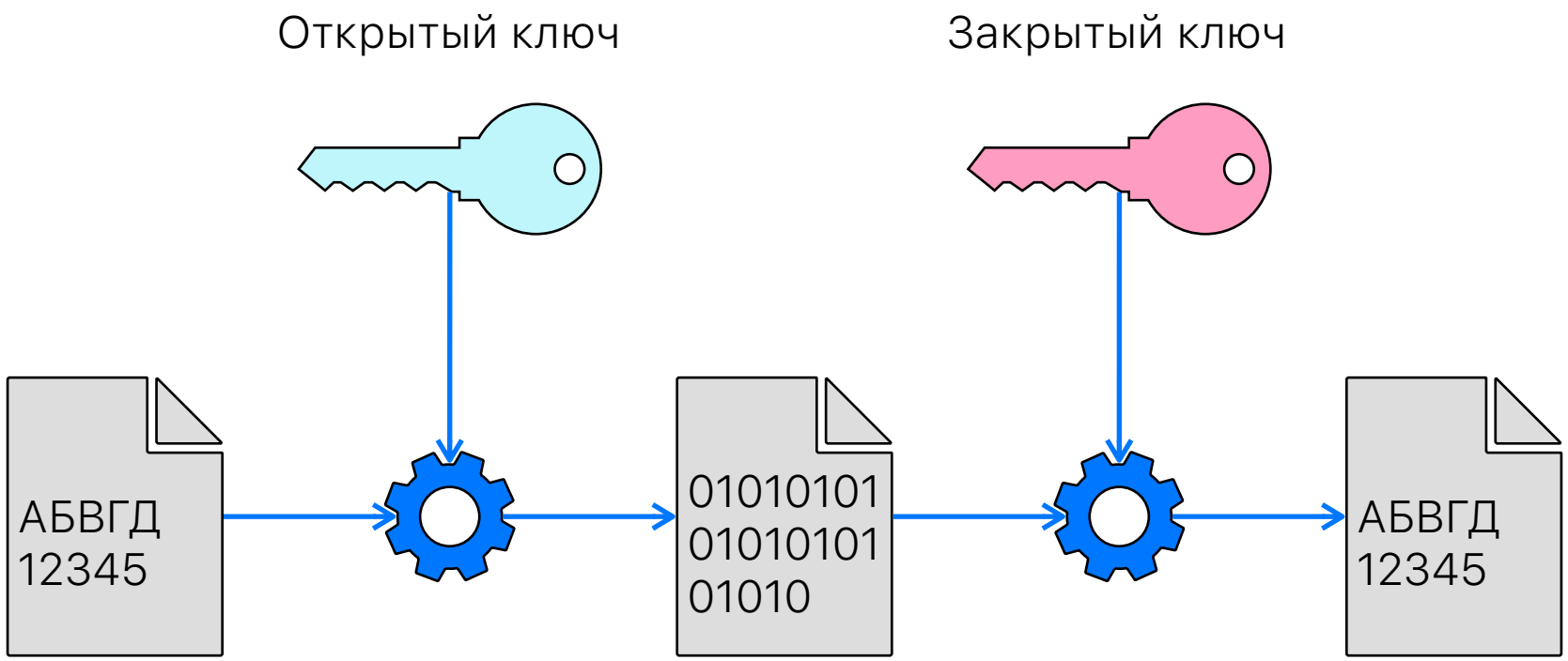
- Появился в API 24. Стал обязательным с API 29
- Direct Boot
- Разные зоны на диске шифруются разными ключами
- Данные пользователя расшифрованы после включения

Основа основ

Симметричное шифрование



Асимметричное шифрование



Выбираем алгоритм шифрования

1. AES
2. RSA
3. DES
4. 3DES
5. Blowfish
6. Twofish
7. ECC
8. RC4
9. IDEA
10. Serpent
11. Skipjack
12. CAST-128
13. Camellia
14. GOST
15. SEED
16. Salsa20
17. Chacha20
18. RC5
19. RC6
20. MARS
21. Anubis
22. Kuznyechik
23. Kalyna
24. ARIA
25. HIGHT
26. XTEA
27. XXTEA
28. Speck
29. Simon
30. ZUC
31. Simeck
32. Threefish
33. LBlock
34. Twine
35. Qarma
36. Ascon
37. Grain
38. Spongent
39. Keccak
40. Whirlpool
41. Tiger
42. Sosemanuk
43. Panama
44. FROG
45. ElGamal
46. Blake2
47. Skein
48. Rabbit

Выбираем алгоритм шифрования

1. AES
2. RSA
3. DES
4. 3DES
5. Blowfish
6. Twofish
7. ECC
8. RC4
9. IDEA
10. Serpent
11. Skipjack
12. CAST-128
13. Camellia
14. GOST
15. SEED
16. Salsa20
17. Chacha20
18. RC5
19. RC6
20. MARS
21. Anubis
22. Kuznyechik
23. Kalyna
24. ARIA
25. HIGHT
26. XTEA
27. XXTEA
28. Speck
29. Simon
30. ZUC
31. Simeck
32. Threefish
33. LBlock
34. Twine
35. Qarma
36. Ascon
37. Grain
38. Spongent
39. Keccak
40. Whirlpool
41. Tiger
42. Sosemanuk
43. Panama
44. FROG
45. ElGamal
46. Blake2
47. Skein
48. Rabbit

Выбираем блочный режим шифрования

1. ECB
2. CBC
3. CFB
4. OFB
5. CTR
6. GCM
7. EAX
8. OCB
9. CCM
10. GCM-SIV
11. PCBC
12. EME
13. IAPM
14. XEX
15. LRW
16. XCB
17. EME2
18. CFB-8
19. SIV
20. CTS

Выбираем блочный режим шифрования

```
@Retention(RetentionPolicy.SOURCE)
@StringDef(prefix = { "BLOCK_MODE_" }, value = {
    BLOCK_MODE_ECB,
    BLOCK_MODE_CBC,
    BLOCK_MODE_CTR,
    BLOCK_MODE_GCM,
    })
public @interface BlockModeEnum {}

    Electronic Codebook (ECB) block mode.
public static final String BLOCK_MODE_ECB = "ECB";

    Cipher Block Chaining (CBC) block mode.
public static final String BLOCK_MODE_CBC = "CBC";

    Counter (CTR) block mode.
public static final String BLOCK_MODE_CTR = "CTR";

    Galois/Counter Mode (GCM) block mode.
public static final String BLOCK_MODE_GCM = "GCM";
```

Выбираем блочный режим шифрования

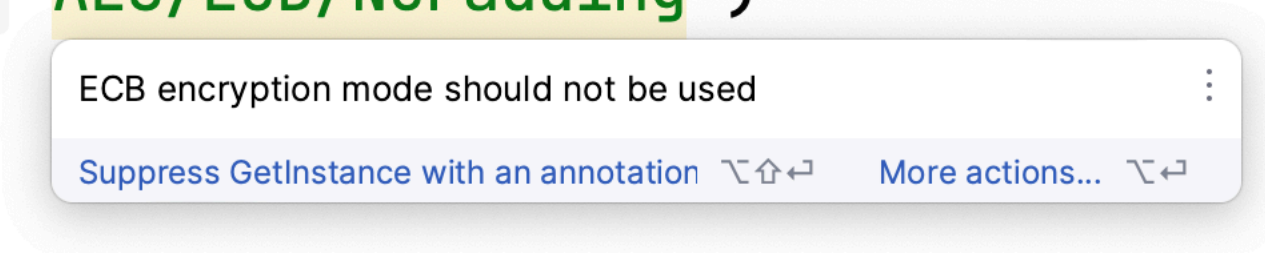
1. ECB
2. CBC
3. CFB
4. OFB
5. CTR
6. GCM
7. EAX
8. OCB
9. CCM
10. GCM-SIV
11. PCBC
12. EME
13. IAPM
14. XEX
15. LRW
16. XCB
17. EME2
18. CFB-8
19. SIV
20. CTS

ECB

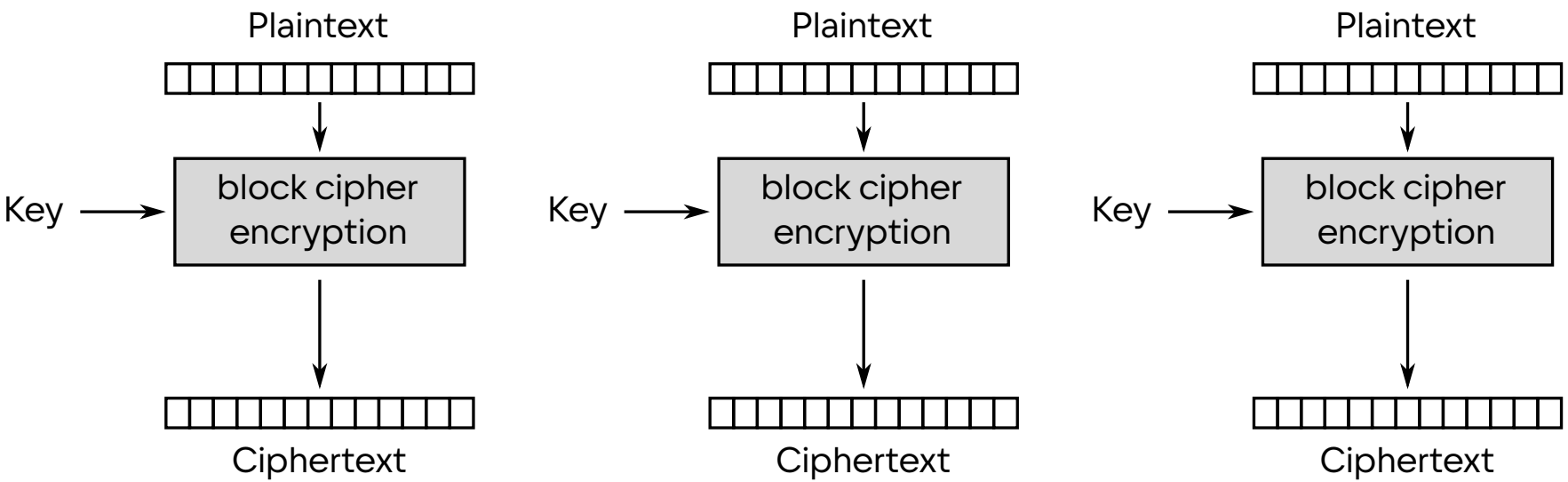
Electronic Code Book

ECB

```
Cipher.getInstance( transformation: "AES/ECB/NoPadding")
```

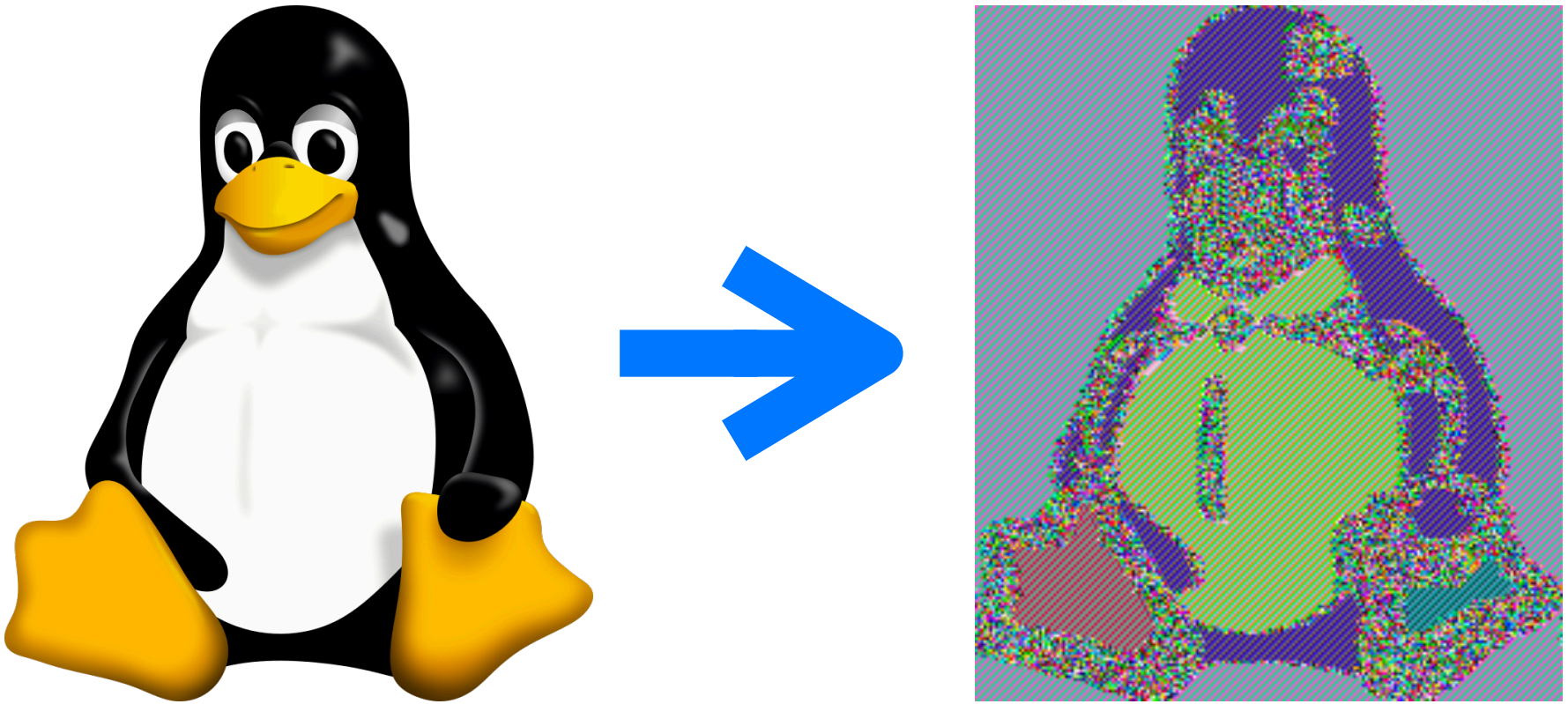


ECB



Electronic Codebook (ECB) mode encryption

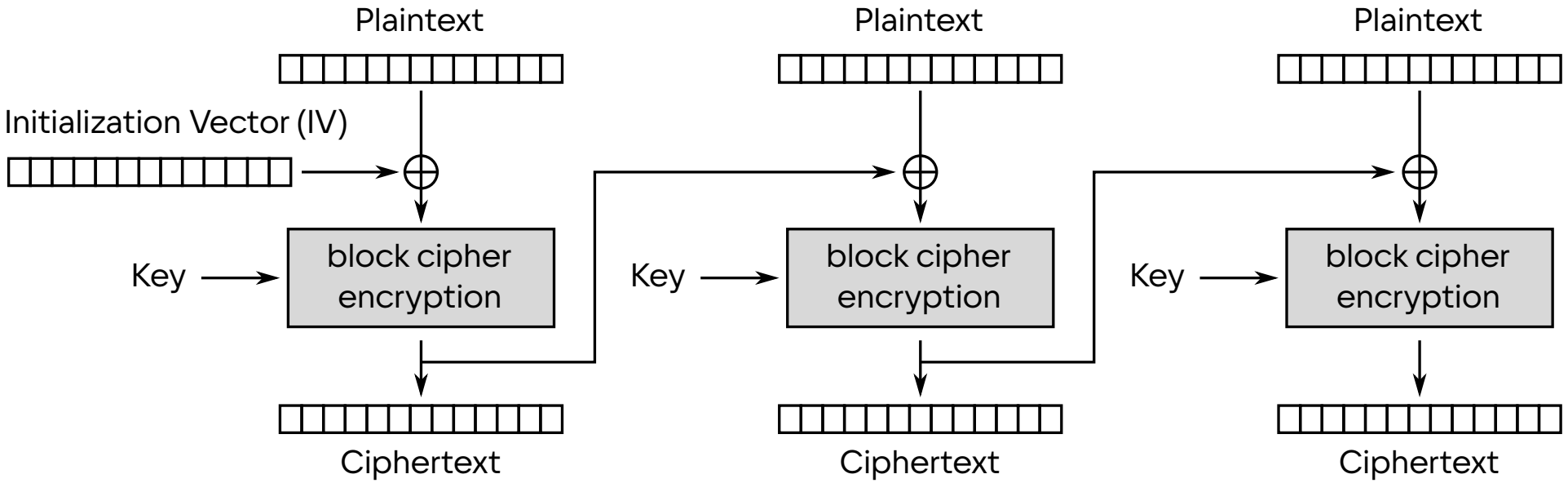
ECB



CBC

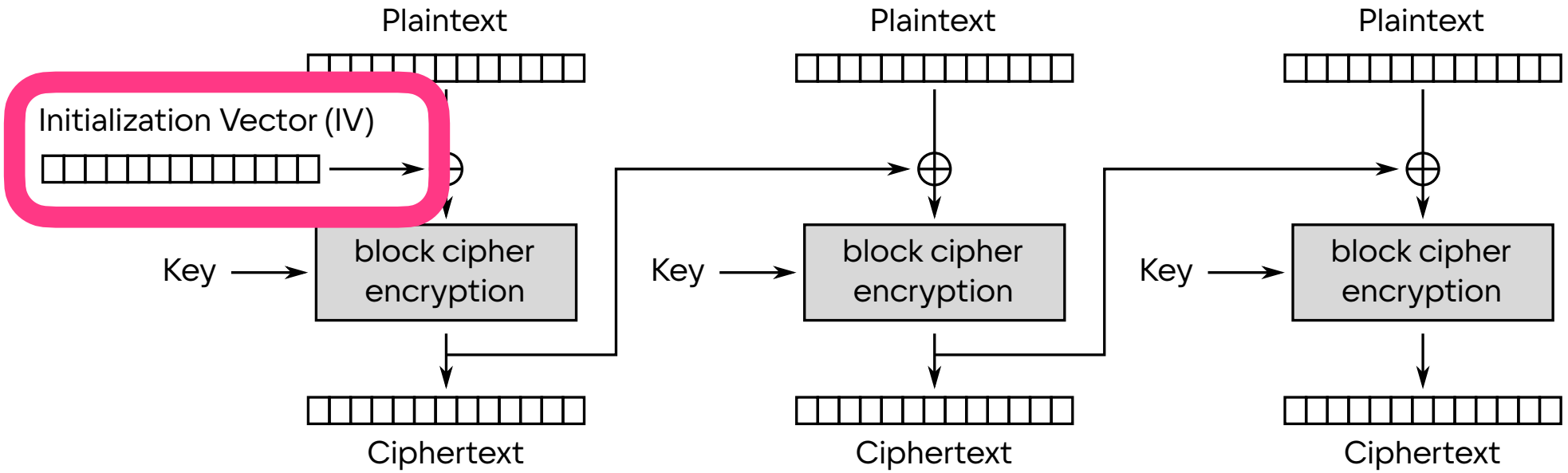
Cipher Block Chaining

CBC



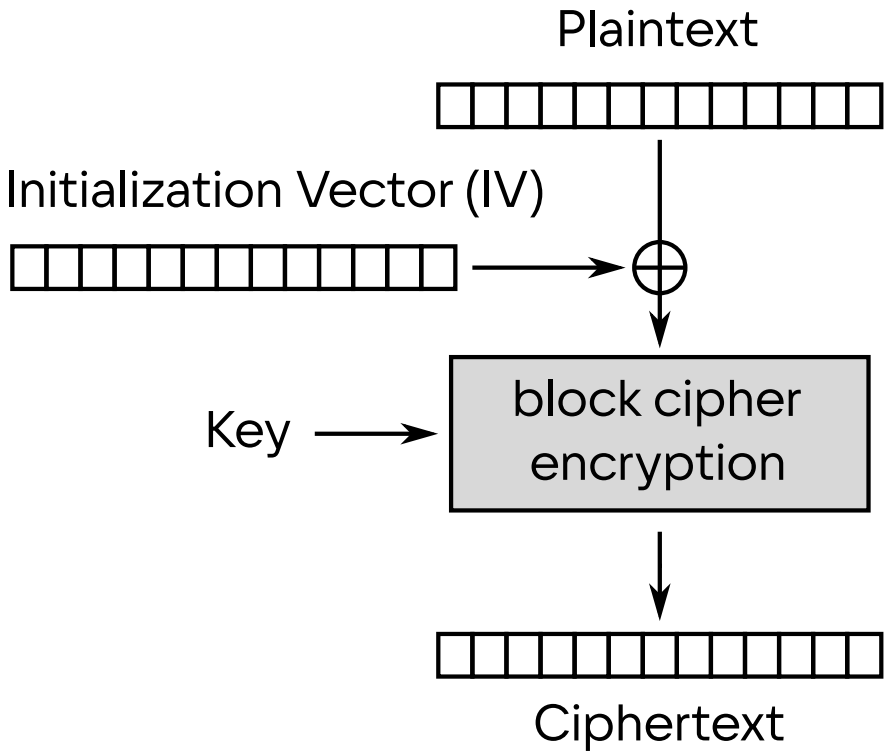
Cipher Block Chaining (CBC) mode encryption

CBC

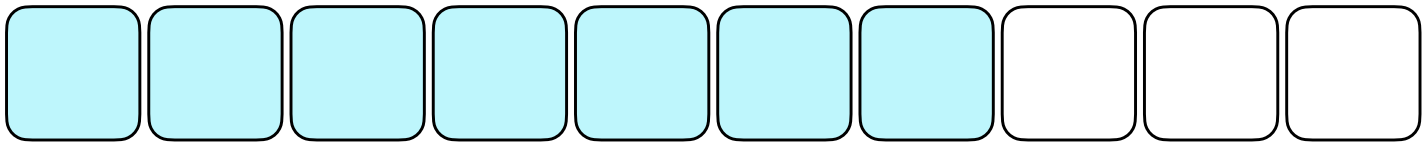


Cipher Block Chaining (CBC) mode encryption

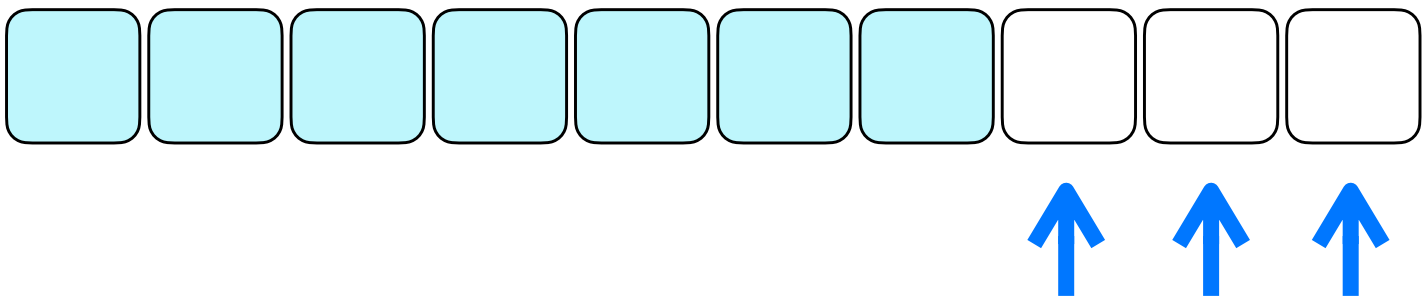
Вектор инициализации



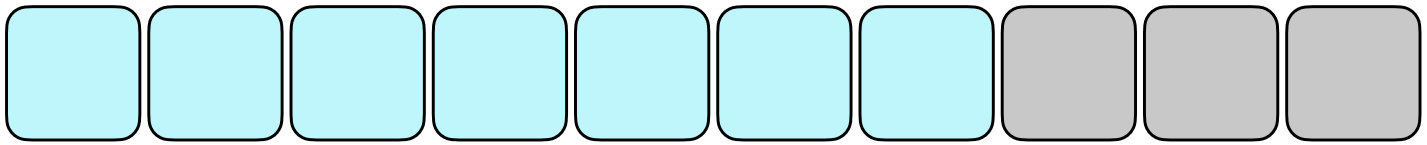
Padding



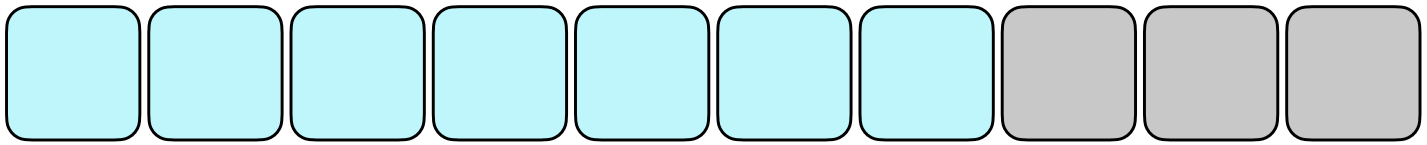
Padding



Padding

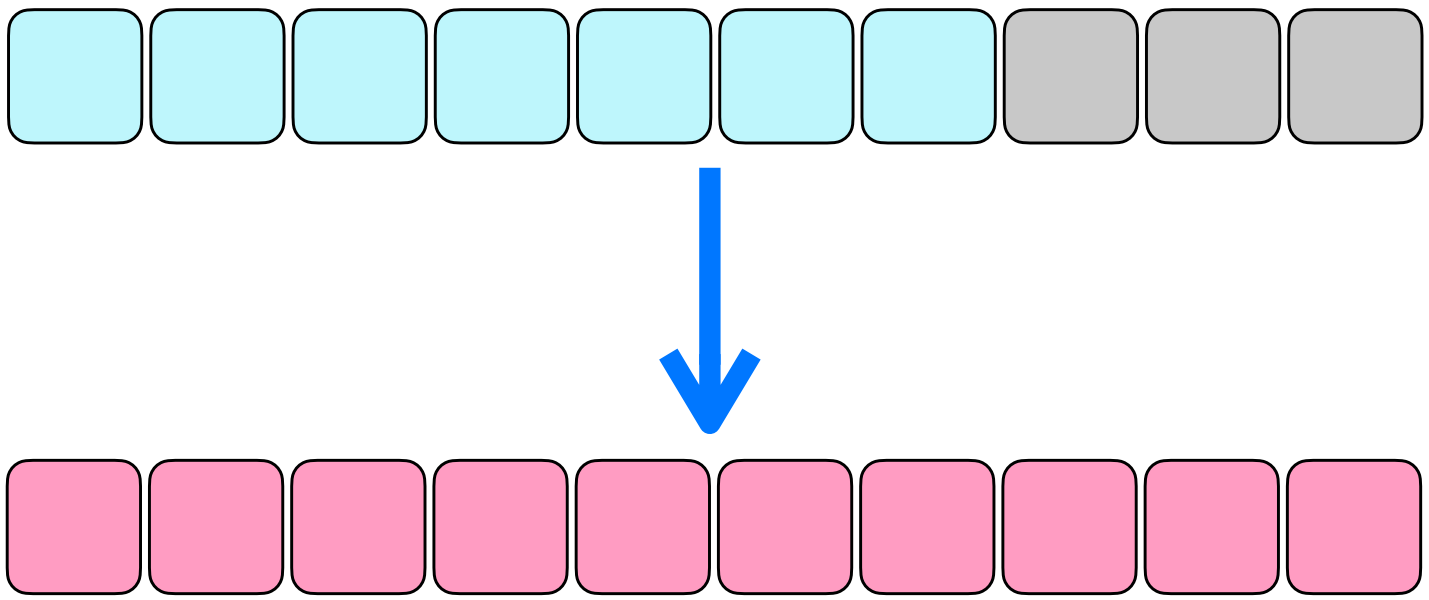


Padding

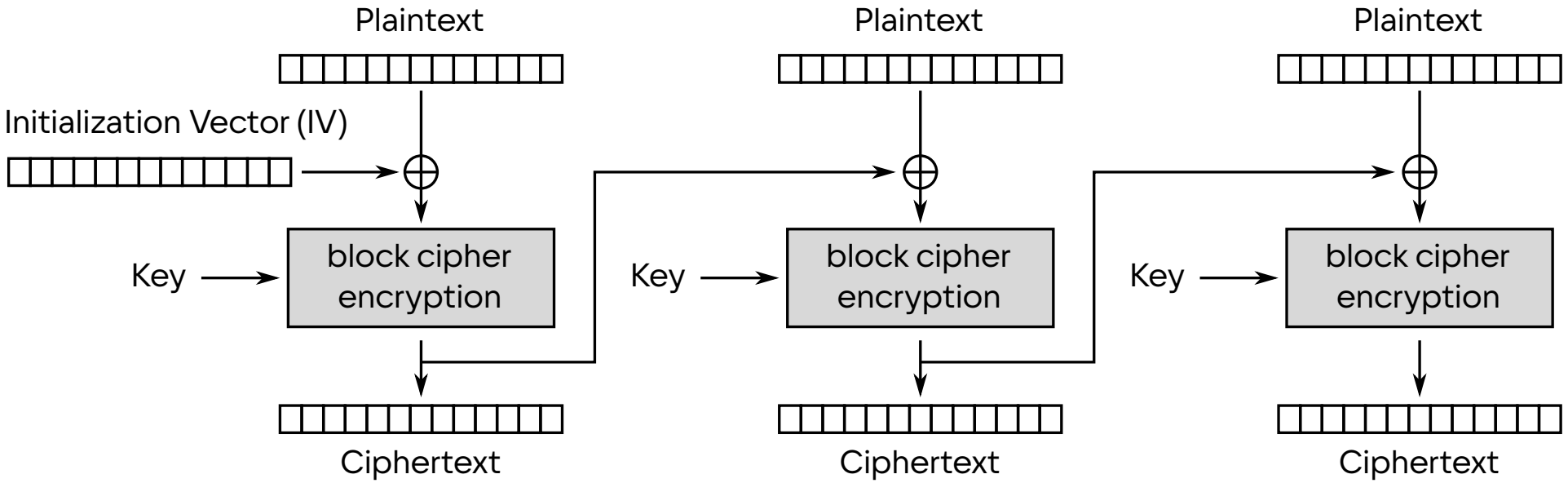


PKCS#7

Padding

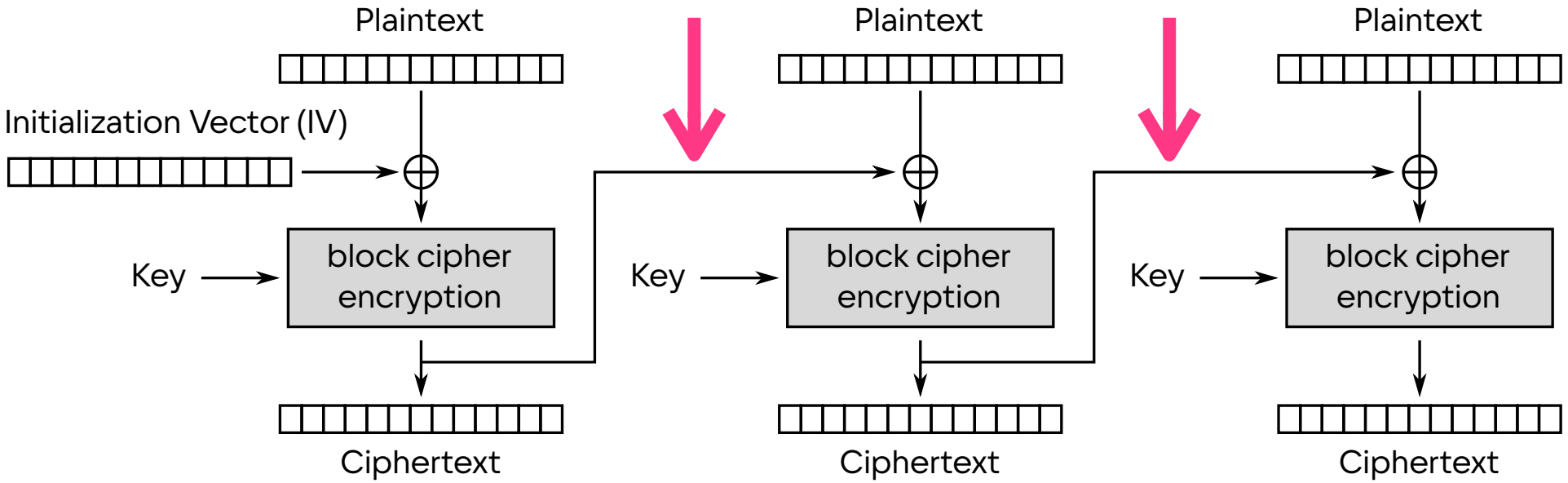


CBC



Cipher Block Chaining (CBC) mode encryption

CBC

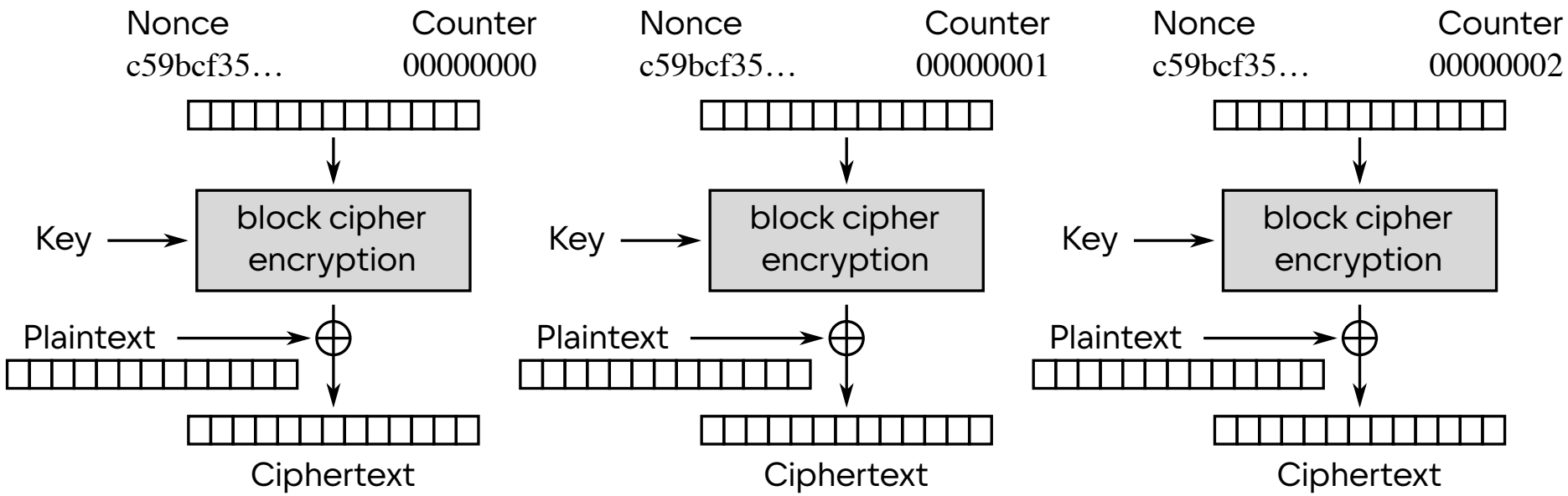


Cipher Block Chaining (CBC) mode encryption

CTR

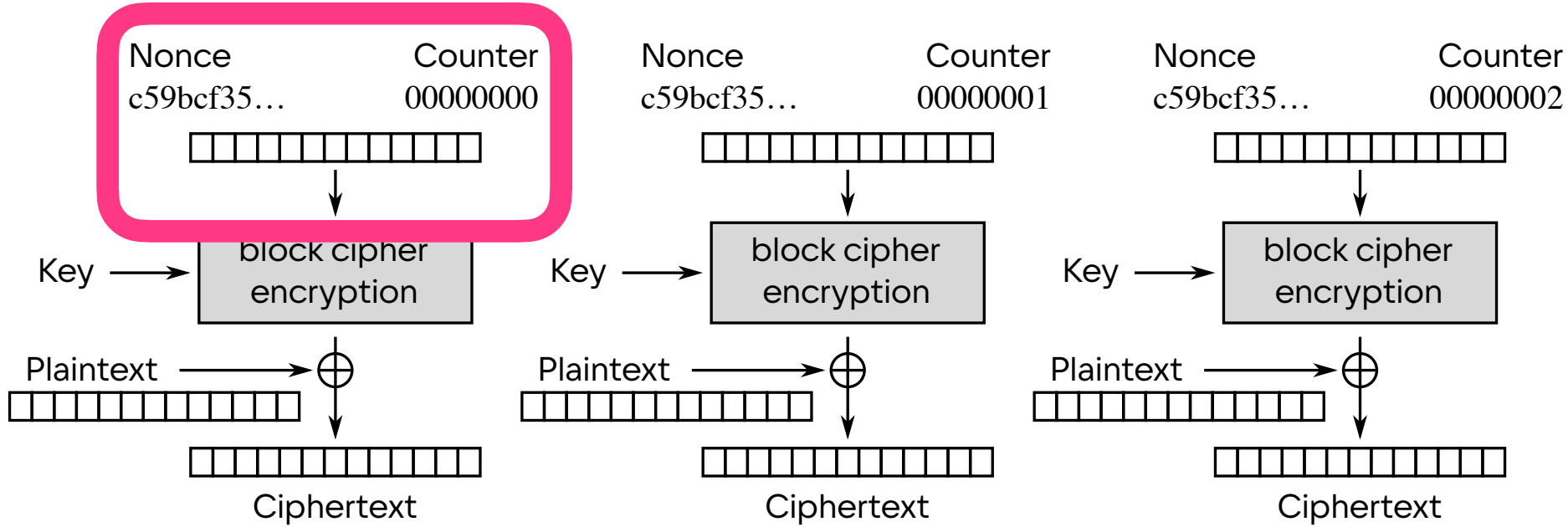
Counter Mode

CTR



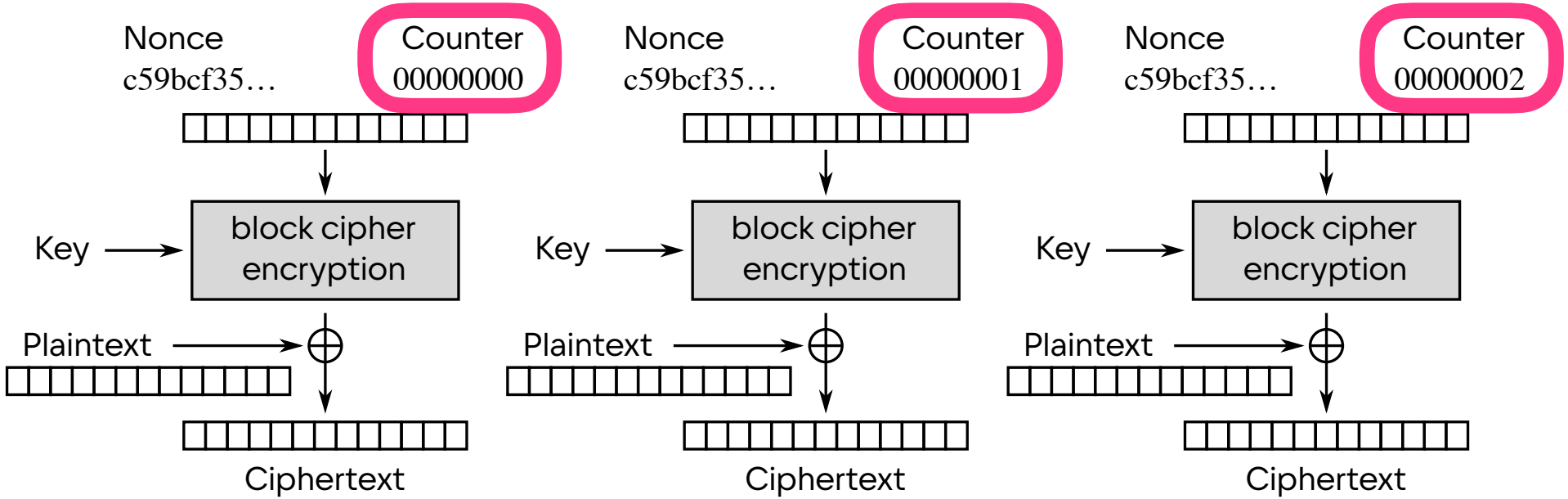
Counter (CTR) mode encryption

CTR



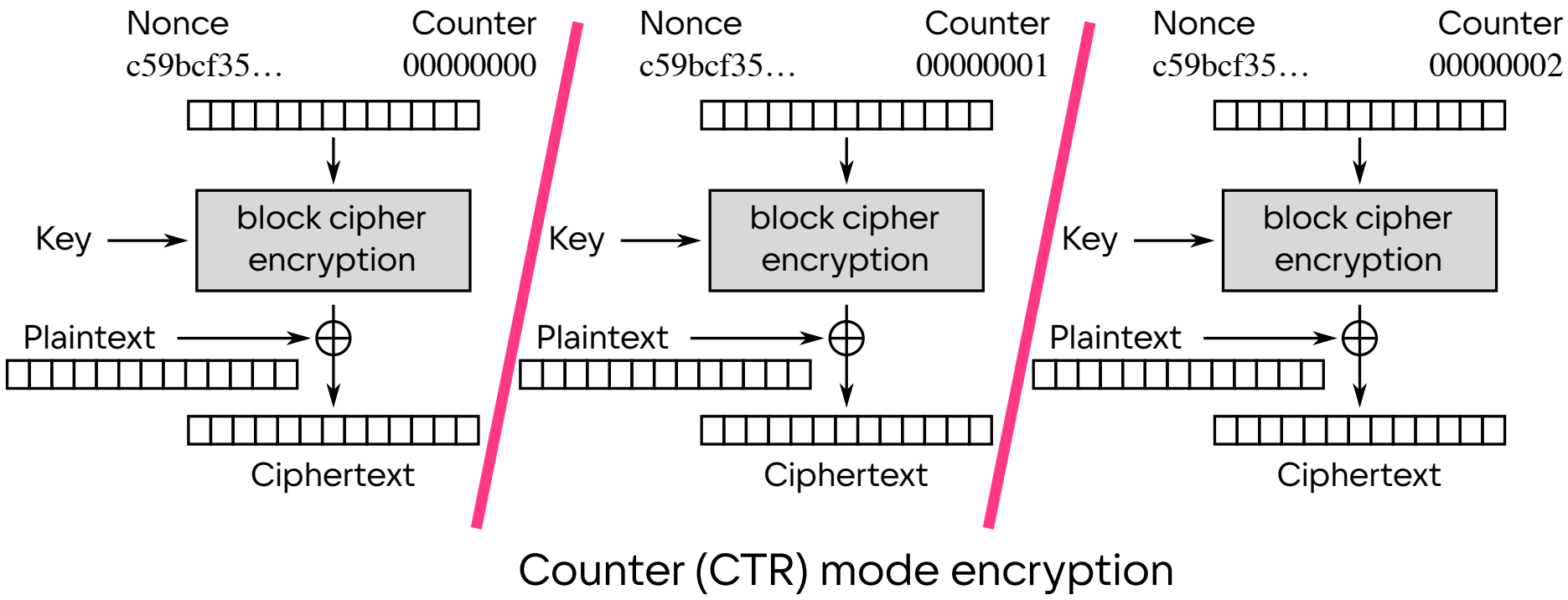
Counter (CTR) mode encryption

CTR



Counter (CTR) mode encryption

CTR

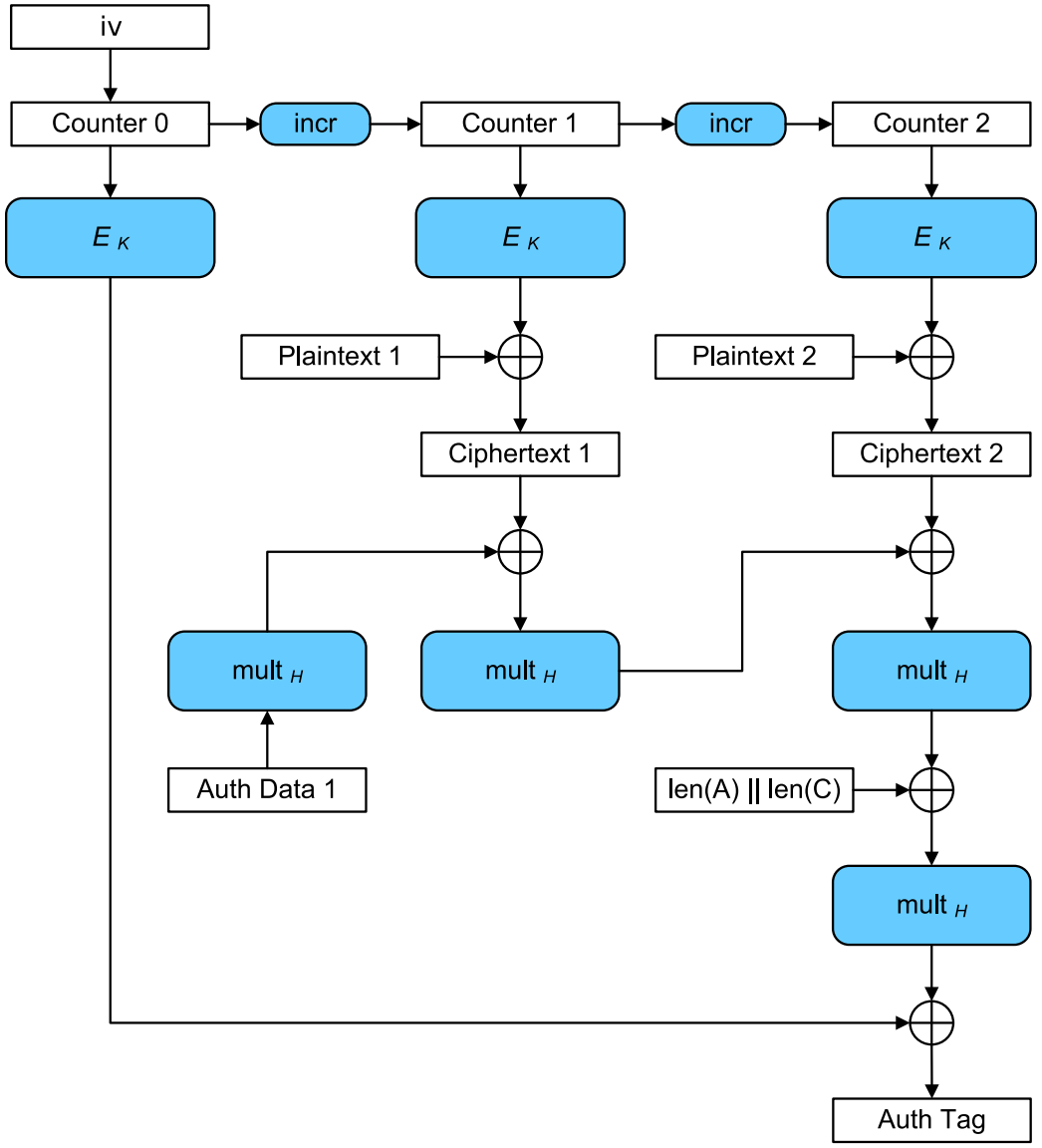


Counter (CTR) mode encryption

GCM

Galois Counter Mode

GCM


iv
Counter 0
incr
Counter 1
incr
Counter 2
E K
E K
E K
Plaintext 1
Plaintext 2
Ciphertext 1
Ciphertext 2
mult H
mult H
mult H
Auth Data 1
len(A) || len(C)
mult H
Auth Tag

GCM = CTR + Auth

Выбираем блочный режим шифрования

1. ECB
2. CBC
3. CFB
4. OFB
5. CTR
6. GCM
7. EAX
8. OCB
9. CCM
10. GCM-SIV
11. PCBC
12. EME
13. IAPM
14. XEX
15. LRW
16. XCB
17. EME2
18. CFB-8
19. SIV
20. CTS

Выбираем блочный режим шифрования

1. ECB
2. CBC
3. CFB
4. OFB
5. CTR
6. GCM
7. EAX
8. OCB
9. CCM
10. GCM-SIV
11. PCBC
12. EME
13. IAPM
14. XEX
15. LRW
16. XCB
17. EME2
18. CFB-8
19. SIV
20. CTS

EncryptedFile

EncryptedFile

```
implementation("androidx.security:security-crypto:1.1.0-alpha06")
```

EncryptedFile

```
private val masterKey = MasterKey.Builder(context)
    .setKeyScheme(MasterKey.KeyScheme.AES256_GCM)
    .build()

val file: File = ...

val encryptedFile = EncryptedFile.Builder(
    context,
    file,
    masterKey,
    EncryptedFile.FileEncryptionScheme.AES256_GCM_HKDF_4KB
).build()

val inputStream: InputStream = encryptedFile.openFileInput()

val outputStream: OutputStream = encryptedFile.openFileOutput()
```

EncryptedFile

```
private val masterKey = MasterKey.Builder(context)
    .setKeyScheme(MasterKey.KeyScheme.AES256_GCM)
    .build()

val file: File = ...

val encryptedFile = EncryptedFile.Builder(
    context,
    file,
    masterKey,
    EncryptedFile.FileEncryptionScheme.AES256_GCM_HKDF_4KB
).build()

val inputStream: InputStream = encryptedFile.openFileInput()

val outputStream: OutputStream = encryptedFile.openFileOutput()
```

EncryptedFile

```
private val masterKey = MasterKey.Builder(context)
    .setKeyScheme(MasterKey.KeyScheme.AES256_GCM)
    .build()

val file: File = ...

val encryptedFile = EncryptedFile.Builder(
    context,
    file,
    masterKey,
    EncryptedFile.FileEncryptionScheme.AES256_GCM_HKDF_4KB
).build()

val inputStream: InputStream = encryptedFile.openFileInput()

val outputStream: OutputStream = encryptedFile.openFileOutput()
```

EncryptedFile

```
private val masterKey = MasterKey.Builder(context)
    .setKeyScheme(MasterKey.KeyScheme.AES256_GCM)
    .build()

val file: File = ...

val encryptedFile = EncryptedFile.Builder(
    context,
    file,
    masterKey,
    EncryptedFile.FileEncryptionScheme.AES256_GCM_HKDF_4KB
).build()

val inputStream: InputStream = encryptedFile.openFileInput()

val outputStream: OutputStream = encryptedFile.openFileOutput()
```

EncryptedFile

```
The encryption scheme to encrypt files.
public enum FileEncryptionScheme {
    The file content is encrypted using StreamingAead with AES-GCM, with the file
    name as associated data.
    For more information please see the Tink documentation:
    AesGcmHkdfStreamingKeyManager ↗ .aes256GcmHkdf4KBTemplate()
    AES256_GCM_HKDF_4KB( keyTemplateName: "AES256_GCM_HKDF_4KB");

    private final String mKeyTemplateName;

    FileEncryptionScheme(String keyTemplateName) { mKeyTemplateName = keyTemplateName; }

    KeyTemplate getKeyTemplate() throws GeneralSecurityException {
        return KeyTemplates.get(mKeyTemplateName);
    }
}
```

EncryptedFile

```
The encryption scheme to encrypt files.
public enum FileEncryptionScheme {
    The file content is encrypted using StreamingAead with AES-GCM, with the file
    name as associated data.
    For more information please see the Tink documentation:
    AesGcmHkdfStreamingKeyManager ↗ .aes256GcmHkdf4KBTemplate()
    AES256_GCM_HKDF_4KB( keyTemplateName: "AES256_GCM_HKDF_4KB");

    private final String mKeyTemplateName;

    FileEncryptionScheme(String keyTemplateName) { mKeyTemplateName = keyTemplateName; }

    KeyTemplate getKeyTemplate() throws GeneralSecurityException {
        return KeyTemplates.get(mKeyTemplateName);
    }
}
```

EncryptedFile

```
implementation("androidx.security:security-crypto:1.1.0-alpha06")
```

Google (14)

	Version	Vulnerabilities	Repository	Usages	Date
1.1.x	1.1.0-alpha06		Google	56	*Apr 19, 2023*
	1.1.0-alpha05		Google	14	*Feb 22, 2023*
	1.1.0-alpha04		Google	16	*Nov 09, 2022*
	1.1.0-alpha03		Google	80	*Dec 08, 2020*
	1.1.0-alpha02		Google	16	*Aug 07, 2020*
	1.1.0-alpha01		Google	21	*Jun 12, 2020*
	1.0.0		Google	77	*Apr 22, 2021*

Создание ключа

Создание ключа

```
import java.security.Key
import java.util.Random
import javax.crypto.spec.SecretKeySpec

val random = Random()
val key = generateKey("AES", 256)

fun generateKey(cipher: String, keySize: Int): Key {
    val byteArray = ByteArray(keySize / 8)
    random.nextBytes(byteArray)
    return SecretKeySpec(byteArray, cipher)
}
```

Создание ключа

```
import java.security.Key
import java.util.Random
import javax.crypto.spec.SecretKeySpec

val random = Random()
val key = generateKey("AES", 256)

fun generateKey(cipher: String, keySize: Int): Key {
    val byteArray = ByteArray(keySize / 8)
    random.nextBytes(byteArray)
    return SecretKeySpec(byteArray, cipher)
}
```

Создание ключа

```
import java.security.Key
import java.util.Random
import javax.crypto.spec.SecretKeySpec

val random = Random()
val key = generateKey("AES", 256)

fun generateKey(cipher: String, keySize: Int): Key {
    val byteArray = ByteArray(keySize / 8)
    random.nextBytes(byteArray)
    return SecretKeySpec(byteArray, cipher)
}
```

Создание ключа

```
import java.security.Key
import java.util.Random
import javax.crypto.spec.SecretKeySpec

val random = Random()
val key = generateKey("AES", 256)

fun generateKey(cipher: String, keySize: Int): Key {
    val byteArray = ByteArray(keySize / 8)
    random.nextBytes(byteArray)
    return SecretKeySpec(byteArray, cipher)
}
```

Создание ключа

```
import java.security.Key
import java.security.SecureRandom
import javax.crypto.spec.SecretKeySpec

val random = SecureRandom()
val key = generateKey("AES", 256)

fun generateKey(cipher: String, keySize: Int): Key {
    val byteArray = ByteArray(keySize / 8)
    random.nextBytes(byteArray)
    return SecretKeySpec(byteArray, cipher)
}
```

KeyGenerator

Random/SecureRandom

KeyGenerator

```
private fun generateKey(): Key {
    val keyGenerator = KeyGenerator.getInstance("AES", "AndroidKeyStore")
    val spec = createSpec()
    keyGenerator.init(spec)
    return keyGenerator.generateKey()
}

private fun createSpec(): KeyGenParameterSpec {
    val purposes = KeyProperties.PURPOSE_ENCRYPT or KeyProperties.PURPOSE_DECRYPT
    return KeyGenParameterSpec.Builder("MyPerfectKey", purposes)
        .setKeySize(256)
        .setBlockModes(KeyProperties.BLOCK_MODE_CTR)
        .setEncryptionPaddings(KeyProperties.ENCRYPTION_PADDING_NONE)
        .build()
}
```

KeyGenerator

```
private fun generateKey(): Key {
    val keyGenerator = KeyGenerator.getInstance("AES", "AndroidKeyStore")
    val spec = createSpec()
    keyGenerator.init(spec)
    return keyGenerator.generateKey()
}

private fun createSpec(): KeyGenParameterSpec {
    val purposes = KeyProperties.PURPOSE_ENCRYPT or KeyProperties.PURPOSE_DECRYPT
    return KeyGenParameterSpec.Builder("MyPerfectKey", purposes)
        .setKeySize(256)
        .setBlockModes(KeyProperties.BLOCK_MODE_CTR)
        .setEncryptionPaddings(KeyProperties.ENCRYPTION_PADDING_NONE)
        .build()
}
```

KeyGenerator

```
private fun generateKey(): Key {
    val keyGenerator = KeyGenerator.getInstance("AES", "AndroidKeyStore")
    val spec = createSpec()
    keyGenerator.init(spec)
    return keyGenerator.generateKey()
}

private fun createSpec(): KeyGenParameterSpec {
    val purposes = KeyProperties.PURPOSE_ENCRYPT or KeyProperties.PURPOSE_DECRYPT
    return KeyGenParameterSpec.Builder("MyPerfectKey", purposes)
        .setKeySize(256)
        .setBlockModes(KeyProperties.BLOCK_MODE_CTR)
        .setEncryptionPaddings(KeyProperties.ENCRYPTION_PADDING_NONE)
        .build()
}
```

KeyGenerator

```
private fun generateKey(): Key {
    val keyGenerator = KeyGenerator.getInstance("AES", "AndroidKeyStore")
    val spec = createSpec()
    keyGenerator.init(spec)
    return keyGenerator.generateKey()
}

private fun createSpec(): KeyGenParameterSpec {
    val purposes = KeyProperties.PURPOSE_ENCRYPT or KeyProperties.PURPOSE_DECRYPT
    return KeyGenParameterSpec.Builder("MyPerfectKey", purposes)
        .setKeySize(256)
        .setBlockModes(KeyProperties.BLOCK_MODE_CTR)
        .setEncryptionPaddings(KeyProperties.ENCRYPTION_PADDING_NONE)
        .build()
}
```

KeyGenerator

```
private fun generateKey(): Key {
    val keyGenerator = KeyGenerator.getInstance("AES", "AndroidKeyStore")
    val spec = createSpec()
    keyGenerator.init(spec)
    return keyGenerator.generateKey()
}

private fun createSpec(): KeyGenParameterSpec {
    val purposes = KeyProperties.PURPOSE_ENCRYPT or KeyProperties.PURPOSE_DECRYPT
    return KeyGenParameterSpec.Builder("MyPerfectKey", purposes)
        .setKeySize(256)
        .setBlockModes(KeyProperties.BLOCK_MODE_CTR)
        .setEncryptionPaddings(KeyProperties.ENCRYPTION_PADDING_NONE)
        .build()
}
```

Создание ключа до API 23

KeyStore до API 23 не умеет хранить симметричные ключи

Создание ключа до API 23

1) С помощью KeyGenerator генерируем пару ключей

Создание ключа до API 23

1) С помощью KeyGenerator генерируем пару ключей

2) Генерируем симметричный ключ

Создание ключа до API 23

1) С помощью KeyGenerator генерируем пару ключей
2) Генерируем симметричный ключ
3) Шифруем симметричный ключ публичным ключом

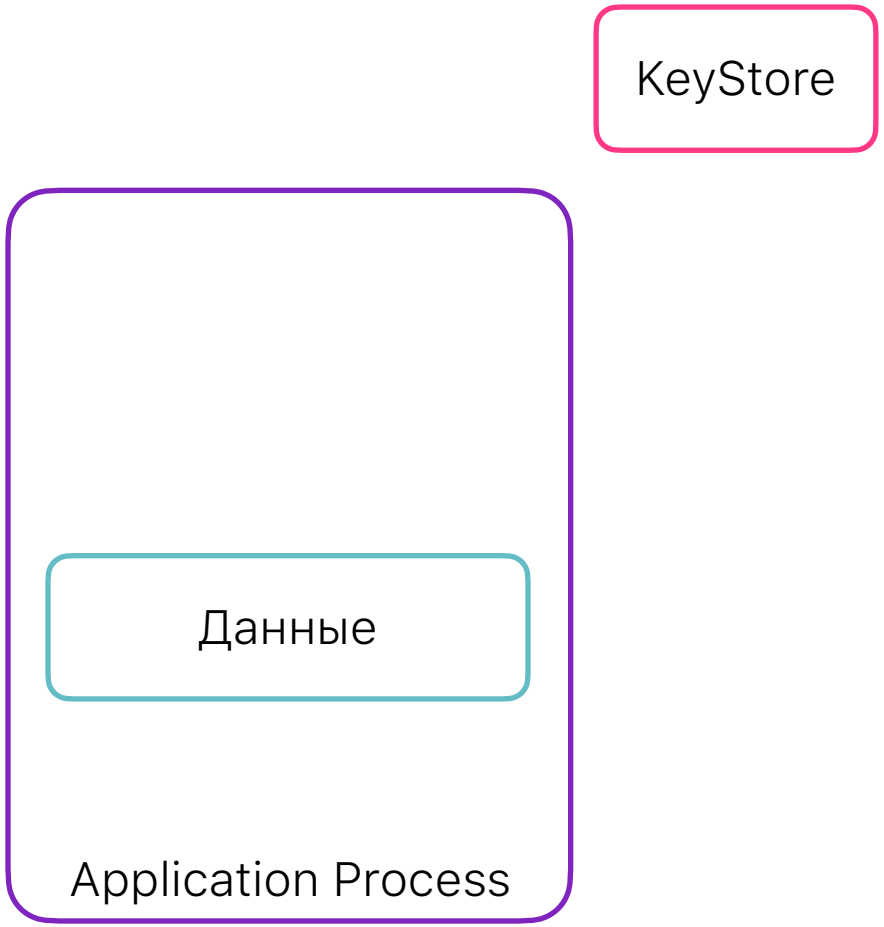
Создание ключа до API 23

1) С помощью KeyGenerator генерируем пару ключей
2) Генерируем симметричный ключ
3) Шифруем симметричный ключ публичным ключом
4) Кладем зашифрованный симметричный ключ в SharedPrefs

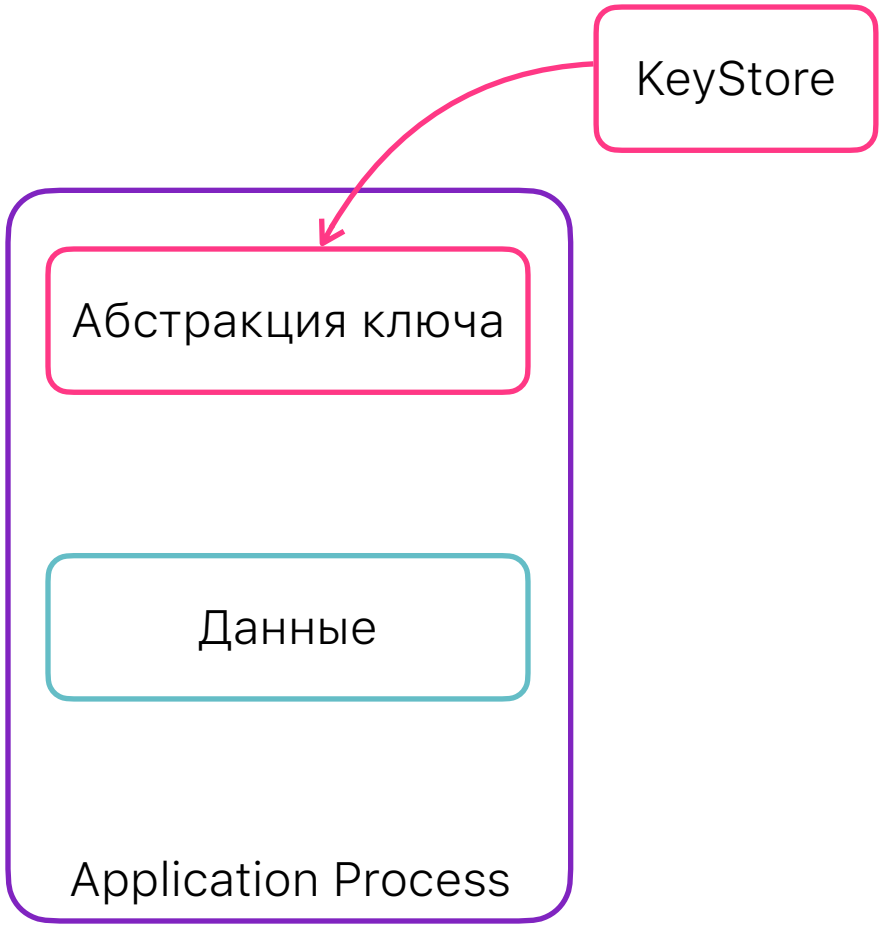
KeyStore

KeyStore - хранилище секретной информации

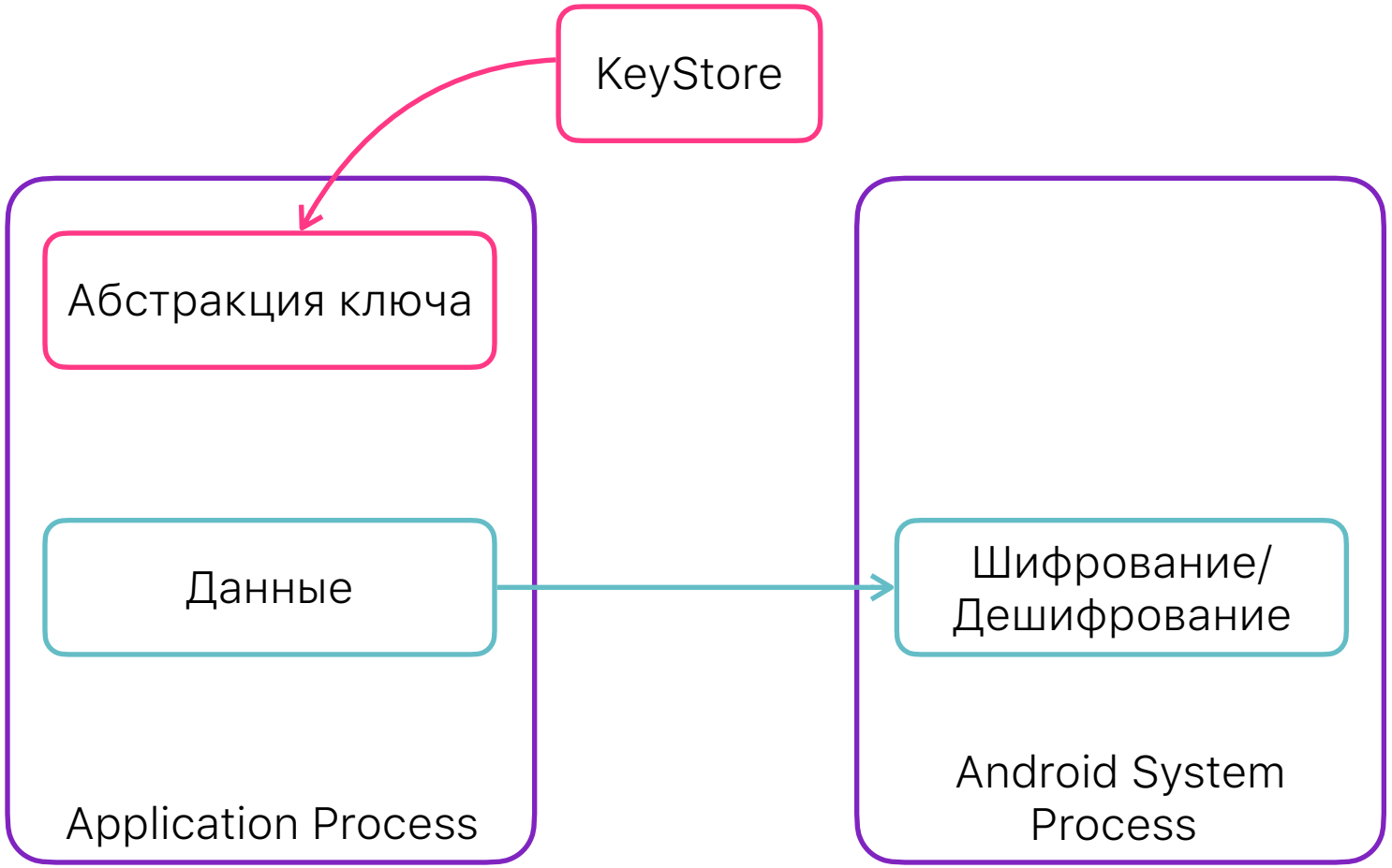
KeyStore



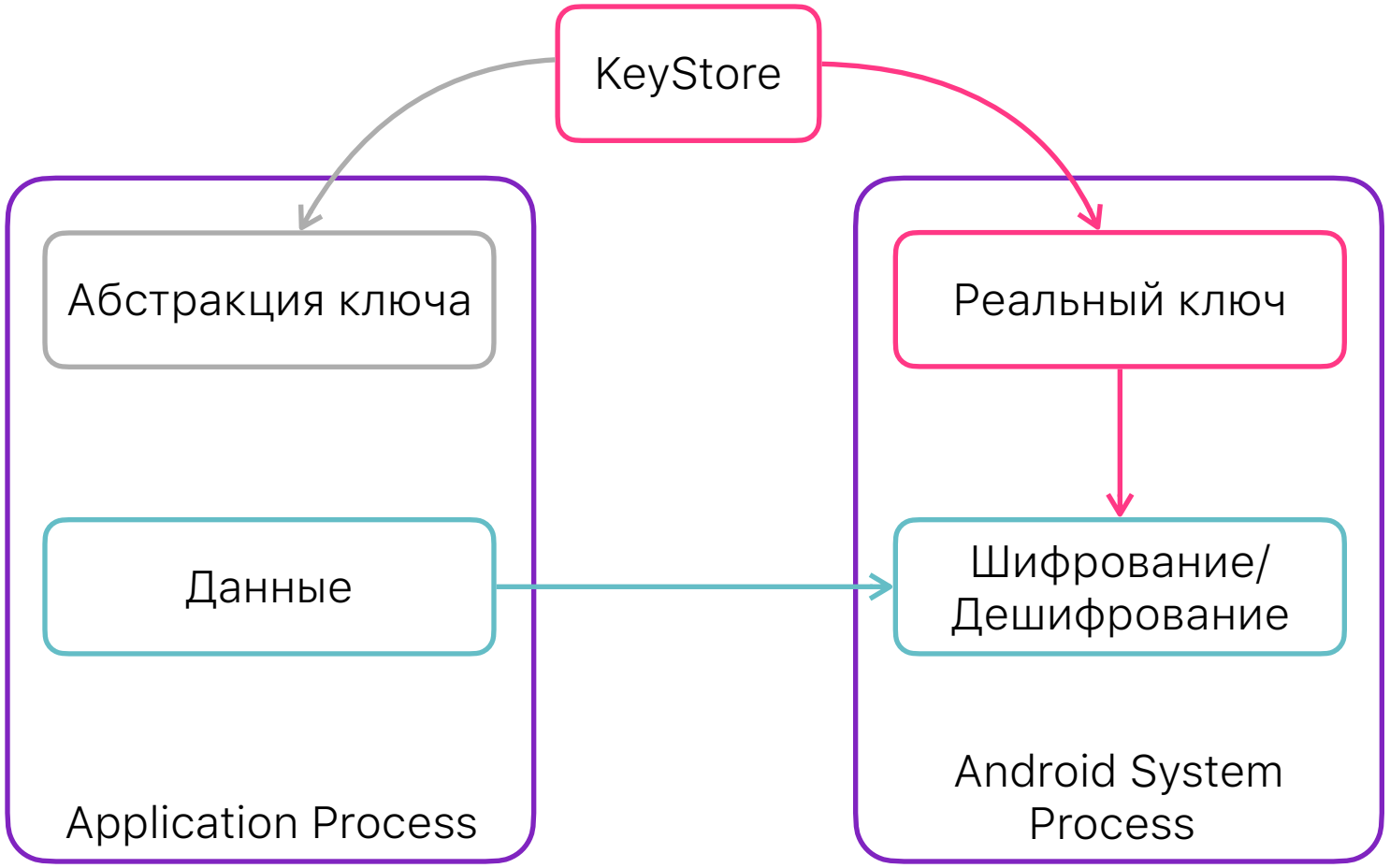
KeyStore



KeyStore


KeyStore
Абстракция ключа
Данные
Application Process
Шифрование/
Дешифрование
Android System
Process

KeyStore


KeyStore
Абстракция ключа
Данные
Application Process
Реальный ключ
Шифрование/
Дешифрование
Android System
Process

KeyStore

AOSP > Docs > Security

Trusty TEE

KeyStore

AOSP > Docs > Security

Trusty TEE

AOSP > Документы > Безопасность

Надежный ТРОЙНИК

KeyStore

```
val keyGenerator = KeyGenerator.getInstance("AES", "AndroidKeyStore")
val spec = KeyGenParameterSpec.Builder("MyPerfectKey", purposes)
    ...
    .build()
```

```
if (keyStore.containsAlias("MyPerfectKey")) {
    val entry = keyStore.getEntry("MyPerfectKey", null)
    return (entry as KeyStore.SecretKeyEntry).secretKey
}
```

Cipher

Cipher

```
private fun createCipher(key: Key): Cipher {
    val cipher = Cipher.getInstance("AES/CTR/NoPadding")
    val iv = generateIv(16)
    val spec = IvParameterSpec(iv)
    cipher.init(Cipher.ENCRYPT_MODE, key, spec)
    return cipher
}

private fun generateIv(size: Int): ByteArray {
    val array = ByteArray(size)
    SecureRandom().nextBytes(array)
    return array
}
```

Cipher

```
private fun createCipher(key: Key): Cipher {
    val cipher = Cipher.getInstance("AES/CTR/NoPadding")
    val iv = generateIv(16)
    val spec = IvParameterSpec(iv)
    cipher.init(Cipher.ENCRYPT_MODE, key, spec)
    return cipher
}

private fun generateIv(size: Int): ByteArray {
    val array = ByteArray(size)
    SecureRandom().nextBytes(array)
    return array
}
```

Cipher

```
private fun createCipher(key: Key): Cipher {
    val cipher = Cipher.getInstance("AES/CTR/NoPadding")
    val iv = generateIv(16)
    val spec = IvParameterSpec(iv)
    cipher.init(Cipher.ENCRYPT_MODE, key, spec)
    return cipher
}

private fun generateIv(size: Int): ByteArray {
    val array = ByteArray(size)
    SecureRandom().nextBytes(array)
    return array
}
```

Cipher

```
private fun createCipher(key: Key): Cipher {
    val cipher = Cipher.getInstance("AES/CTR/NoPadding")
    val iv = generateIv(16)
    val spec = IvParameterSpec(iv)
    cipher.init(Cipher.ENCRYPT_MODE, key, spec)
    return cipher
}

private fun generateIv(size: Int): ByteArray {
    val array = ByteArray(size)
    SecureRandom().nextBytes(array)
    return array
}
```

Cipher

```
private fun createCipher(key: Key): Cipher {
    val cipher = Cipher.getInstance("AES/CTR/NoPadding")
    val iv = generateIv(16)
    val spec = IvParameterSpec(iv)
    cipher.init(Cipher.ENCRYPT_MODE, key, spec)
    return cipher
}

private fun generateIv(size: Int): ByteArray {
    val array = ByteArray(size)
    SecureRandom().nextBytes(array)
    return array
}
```

Cipher

```
private fun createCipher(key: Key): Cipher {
    val cipher = Cipher.getInstance("AES/CTR/NoPadding")
    val iv = generateIv(16)
    val spec = IvParameterSpec(iv)
    cipher.init(Cipher.ENCRYPT_MODE, key, spec)
    return cipher
}

private fun generateIv(size: Int): ByteArray {
    val array = ByteArray(size)
    SecureRandom().nextBytes(array)
    return array
}
```

Cipher

```
private fun createCipher(key: Key): Cipher {
    val cipher = Cipher.getInstance("AES/CTR/NoPadding")
    val iv = generateIv(16)
    val spec = IvParameterSpec(iv)
    cipher.init(Cipher.ENCRYPT_MODE, key, spec)
    return cipher
}

private fun generateIv(size: Int): ByteArray {
    val array = ByteArray(size)
    SecureRandom().nextBytes(array)
    return array
}
```

Cipher

```
private fun createCipher(key: Key): Cipher {
    val cipher = Cipher.getInstance("AES/CTR/NoPadding")
    val iv = generateIv(16)
    val spec = IvParameterSpec(iv)
    cipher.init(Cipher.ENCRYPT_MODE, key, spec)
    return cipher
}

private fun generateIv(size: Int): ByteArray {
    val array = ByteArray(size)
    SecureRandom().nextBytes(array)
    return array
}
```

Cipher

```
val buffer = ByteArray(8192)
var bytes = inputStream.read(buffer)
while (bytes >= 0) {
    val decrypted = cipher.update(buffer, 0, bytes)
    if (decrypted.isNotEmpty()) {
        outputStream.write(decrypted, 0, decrypted.size)
    }
    bytes = inputStream.read(buffer)
}

val remaining = cipher.doFinal()
if (remaining.isNotEmpty()) {
    outputStream.write(remaining, 0, remaining.size)
}
```

Cipher

```
val buffer = ByteArray(8192)
var bytes = inputStream.read(buffer)
while (bytes >= 0) {
    val decrypted = cipher.update(buffer, 0, bytes)
    if (decrypted.isNotEmpty()) {
        outputStream.write(decrypted, 0, decrypted.size)
    }
    bytes = inputStream.read(buffer)
}

val remaining = cipher.doFinal()
if (remaining.isNotEmpty()) {
    outputStream.write(remaining, 0, remaining.size)
}
```

Cipher + I/O

```
private fun createCipher(key: Key): Cipher { ... }

fun createInputStream(file: File): InputStream {
    val cipher = createCipher(key)
    return CipherInputStream(file.inputStream(), cipher)
}

fun createOutputStream(file: File): OutputStream {
    val cipher = createCipher(key)
    return CipherOutputStream(file.outputStream(), cipher)
}
```

Cipher + I/O

```
private fun createCipher(key: Key): Cipher { ... }

fun createInputStream(file: File): InputStream {
    val cipher = createCipher(key)
    return CipherInputStream(file.inputStream(), cipher)
}

fun createOutputStream(file: File): OutputStream {
    val cipher = createCipher(key)
    return CipherOutputStream(file.outputStream(), cipher)
}
```

skip()

skip()

```
private fun createCipher(key: Key): Cipher { ... }

fun createInputStream(file: File, offset: Long): InputStream {
    val cipher = createCipher(key)
    val stream = CipherInputStream(file.inputStream(), cipher)
    stream.skip(offset)
    return stream
}
```

skip()

```
private fun createCipher(key: Key): Cipher { ... }

fun createInputStream(file: File, offset: Long): InputStream {
    val cipher = createCipher(key)
    val stream = CipherInputStream(file.inputStream(), cipher)
    stream.skip(offset)
    return stream
}
```

skip() в InputStream

```
public long skip(long n) throws IOException {
    long remaining = n;
    int nr;
    if (n <= 0) return 0;
    int size = (int)Math.min(MAX_SKIP_BUFFER_SIZE, remaining);
    byte[] skipBuffer = new byte[size];
    while (remaining > 0) {
        nr = read(skipBuffer, 0, (int)Math.min(size, remaining));
        if (nr < 0) break;
        remaining -= nr;
    }
    return n - remaining;
}
```

skip() в InputStream

```
public long skip(long n) throws IOException {
    long remaining = n;
    int nr;
    if (n <= 0) return 0;
    int size = (int)Math.min(MAX_SKIP_BUFFER_SIZE, remaining);
    byte[] skipBuffer = new byte[size];
    while (remaining > 0) {
        nr = read(skipBuffer, 0, (int)Math.min(size, remaining));
        if (nr < 0) break;
        remaining -= nr;
    }
    return n - remaining;
}
```

skip() в CipherInputStream

```
private int ostart = 0;
private int ofinish = 0;

public long skip(long n) throws IOException {
    int available = ofinish - ostart;
    if (n > available) {
        n = available;
    }
    if (n < 0) {
        return 0;
    }
    ostart += n;
    return n;
}
```

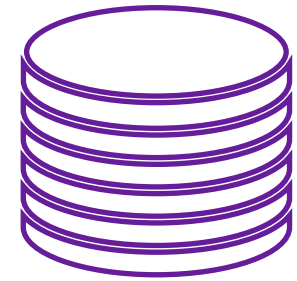
skip() в CipherInputStream

```
private int ostart = 0;
private int ofinish = 0;

public long skip(long n) throws IOException {
    int available = ofinish - ostart;
    if (n > available) {
        n = available;
    }
    if (n < 0) {
        return 0;
    }
    ostart += n;
    return n;
}
```

read() в CipherInputStream

read(N)



Реальные данные

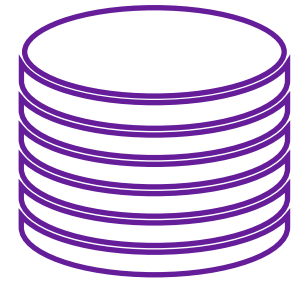
read() в CipherInputStream

read(N)

Cache

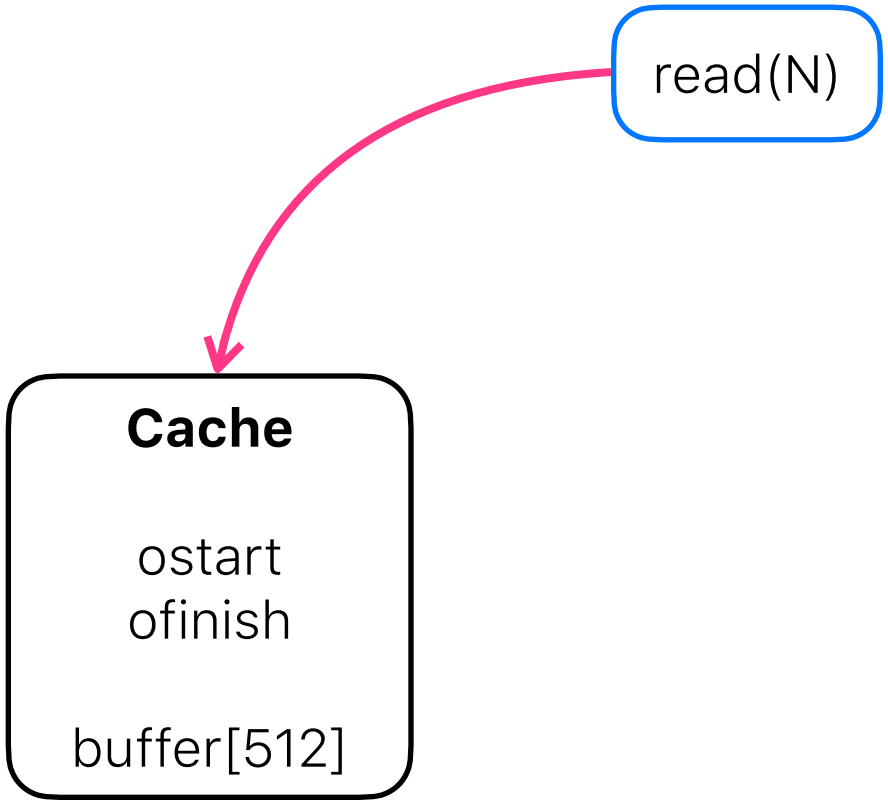
ostart
ofinish

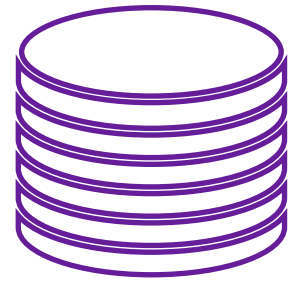
buffer[512]



Реальные данные

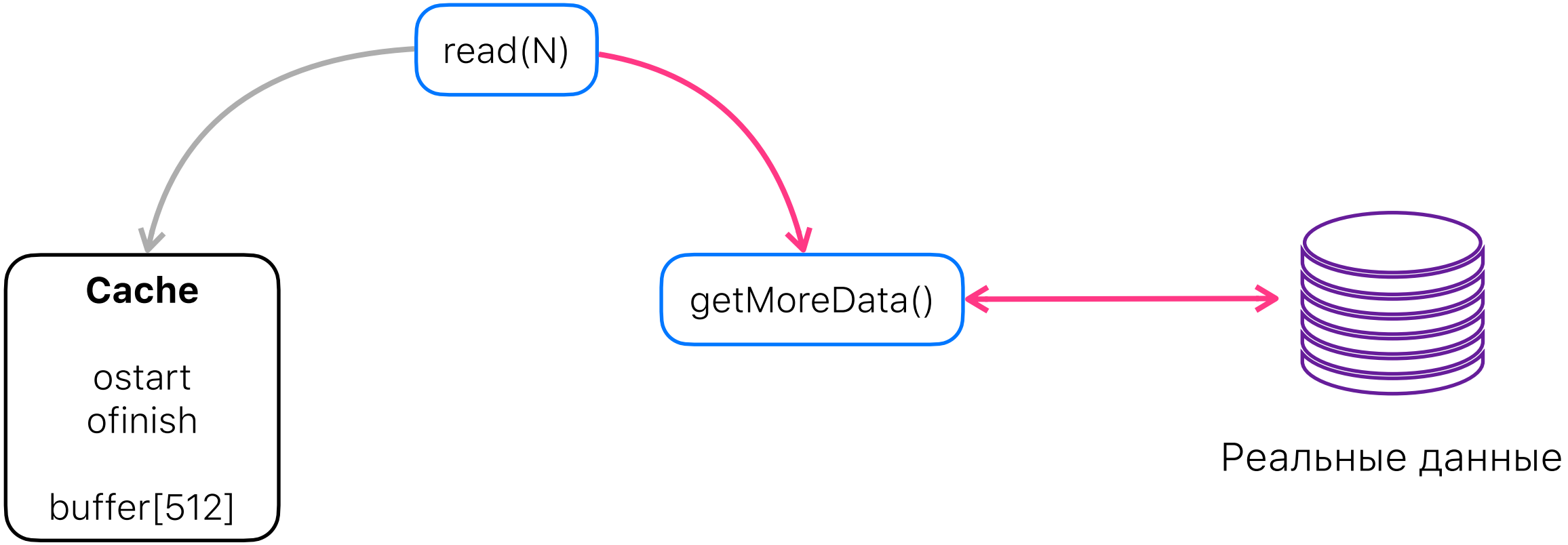
read() в CipherInputStream



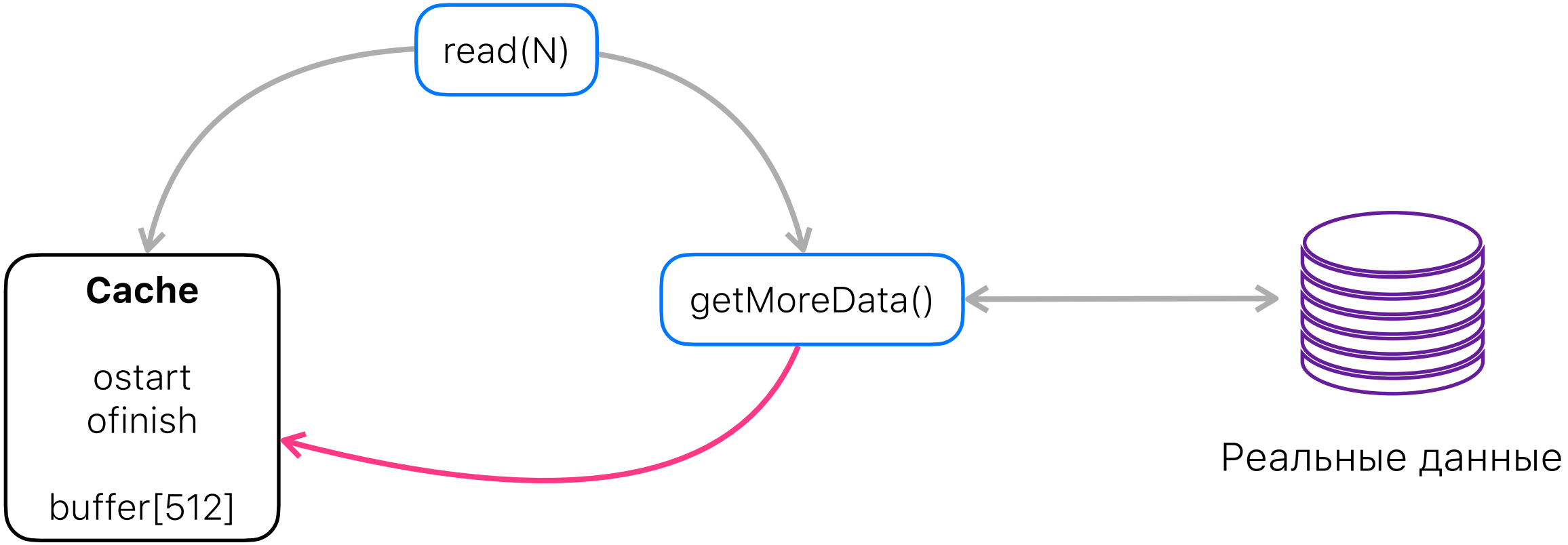


Реальные данные

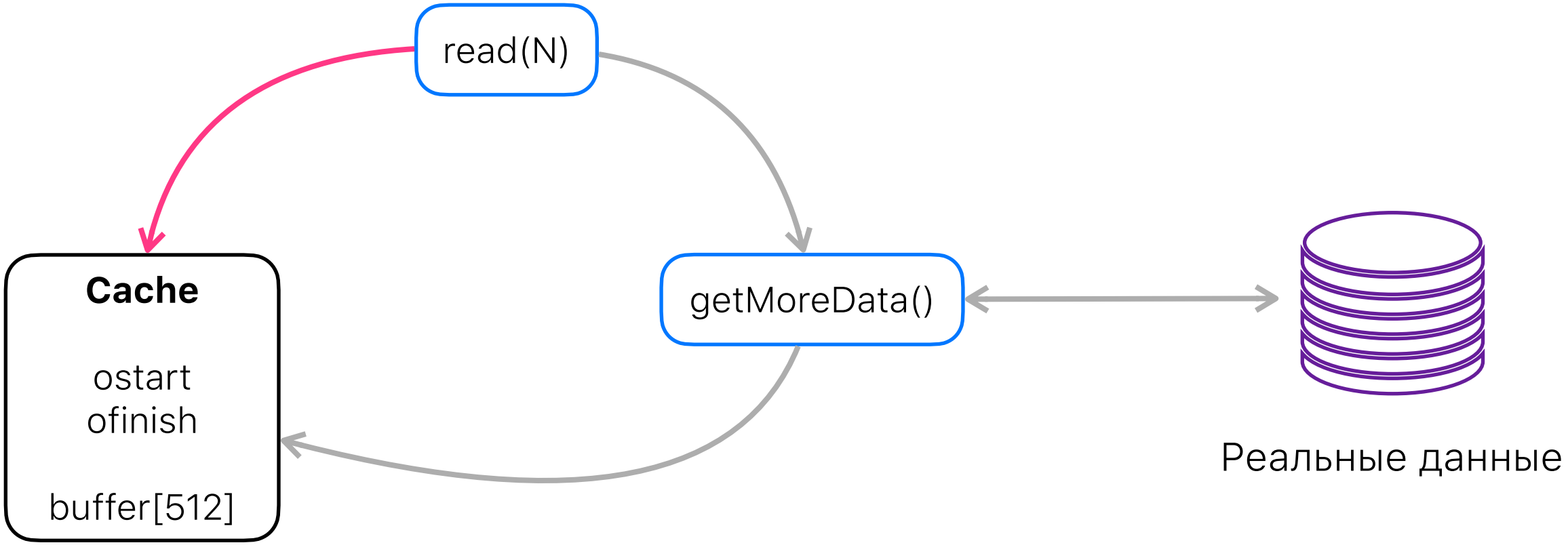
read() в CipherInputStream



read() в CipherInputStream



read() в CipherInputStream



skip(N) + read() ?

~~skip(N) + read() ?~~

skip() 2.0

skip() 2.0

```
private fun generateIv(offset: Long, size: Int = 16): ByteArray {
    val counter = offset / size
    val counterAsBytes = Longs.toByteArray(counter)
    val iv = ByteArray(size)
    ...
    System.arraycopy(
        counterAsBytes,
        0,
        iv,
        iv.size - counterAsBytes.size,
        counterAsBytes.size
    )
    return iv
}
```

skip() 2.0

```
private fun generateIv(offset: Long, size: Int = 16): ByteArray {
    val counter = offset / size
    val counterAsBytes = Longs.toByteArray(counter)
    val iv = ByteArray(size)
    ...
    System.arraycopy(
        counterAsBytes,
        0,
        iv,
        iv.size - counterAsBytes.size,
        counterAsBytes.size
    )
    return iv
}
```

skip() 2.0

```
private fun generateIv(offset: Long, size: Int = 16): ByteArray {
    val counter = offset / size
    val counterAsBytes = Longs.toByteArray(counter)
    val iv = ByteArray(size)
    ...
    System.arraycopy(
        counterAsBytes,
        0,
        iv,
        iv.size - counterAsBytes.size,
        counterAsBytes.size
    )
    return iv
}
```

skip() 2.0

```
fun createInputStream(file: File, offset: Long): InputStream {
    val cipher: Cipher = ...
    val remainder = offset % 16

    val fis = FileInputStream(file)
    fis.skip(offset - remainder)

    val cis = CipherInputStream(fis, cipher)
    customSkip(cis, remainder)
    return cis
}
```

skip() 2.0

```
fun createInputStream(file: File, offset: Long): InputStream {
    val cipher: Cipher = ...
    val remainder = offset % 16

    val fis = FileInputStream(file)
    fis.skip(offset - remainder)

    val cis = CipherInputStream(fis, cipher)
    customSkip(cis, remainder)
    return cis
}
```

skip() 2.0

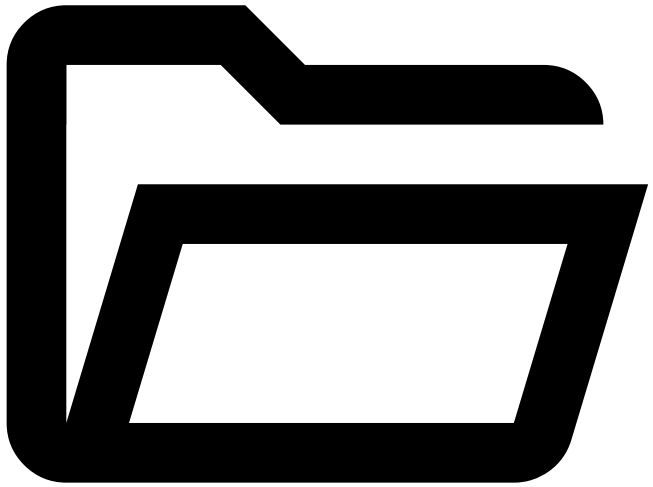
```
fun createInputStream(file: File, offset: Long): InputStream {
    val cipher: Cipher = ...
    val remainder = offset % 16

    val fis = FileInputStream(file)
    fis.skip(offset - remainder)

    val cis = CipherInputStream(fis, cipher)
    customSkip(cis, remainder) <—— skip + read
    return cis
}
```

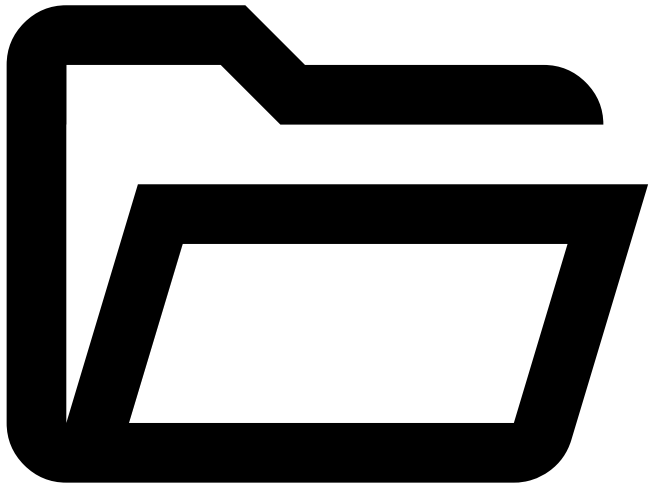
А как шарить?

Шаринг



Downloads

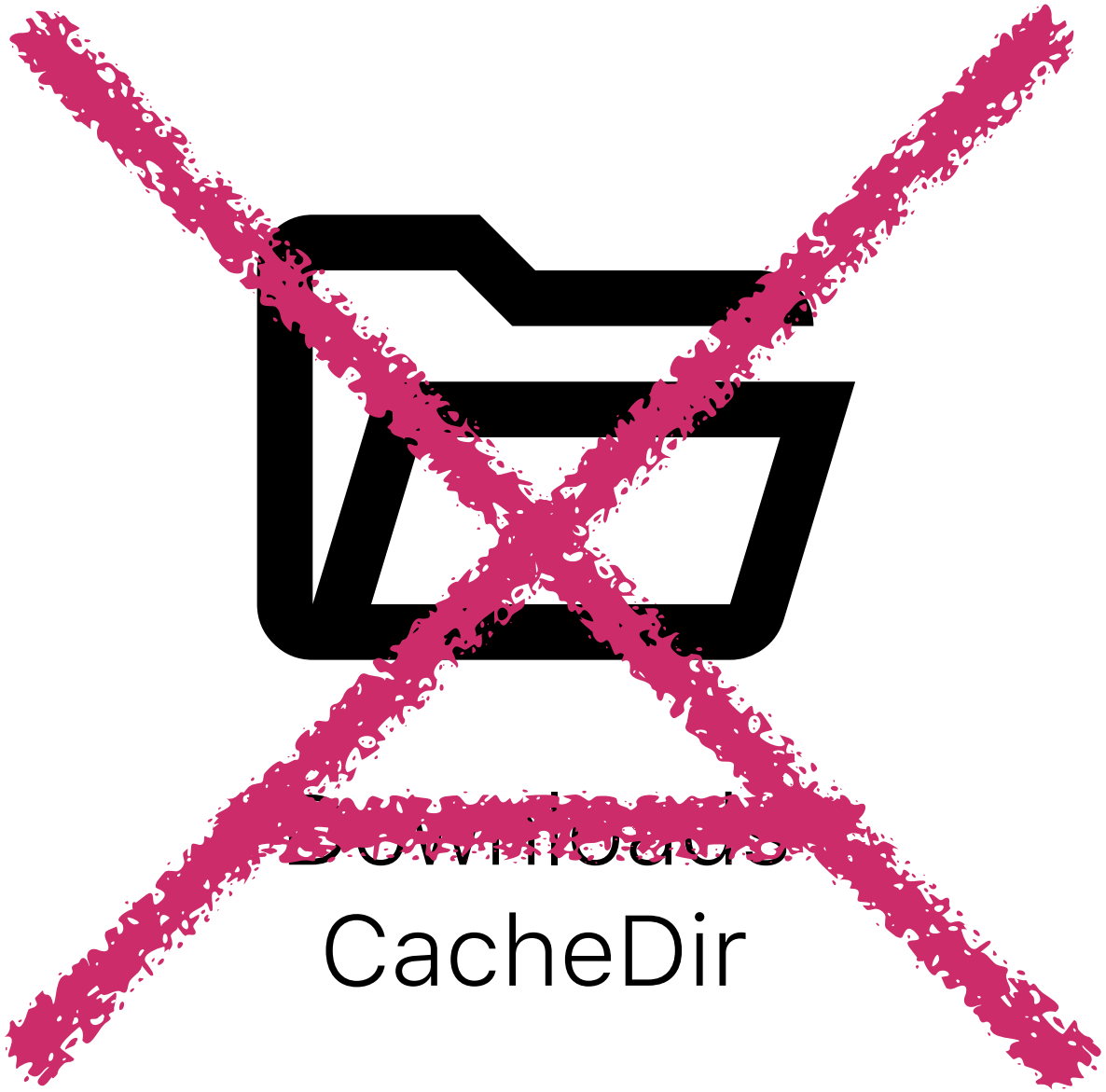
Шаринг

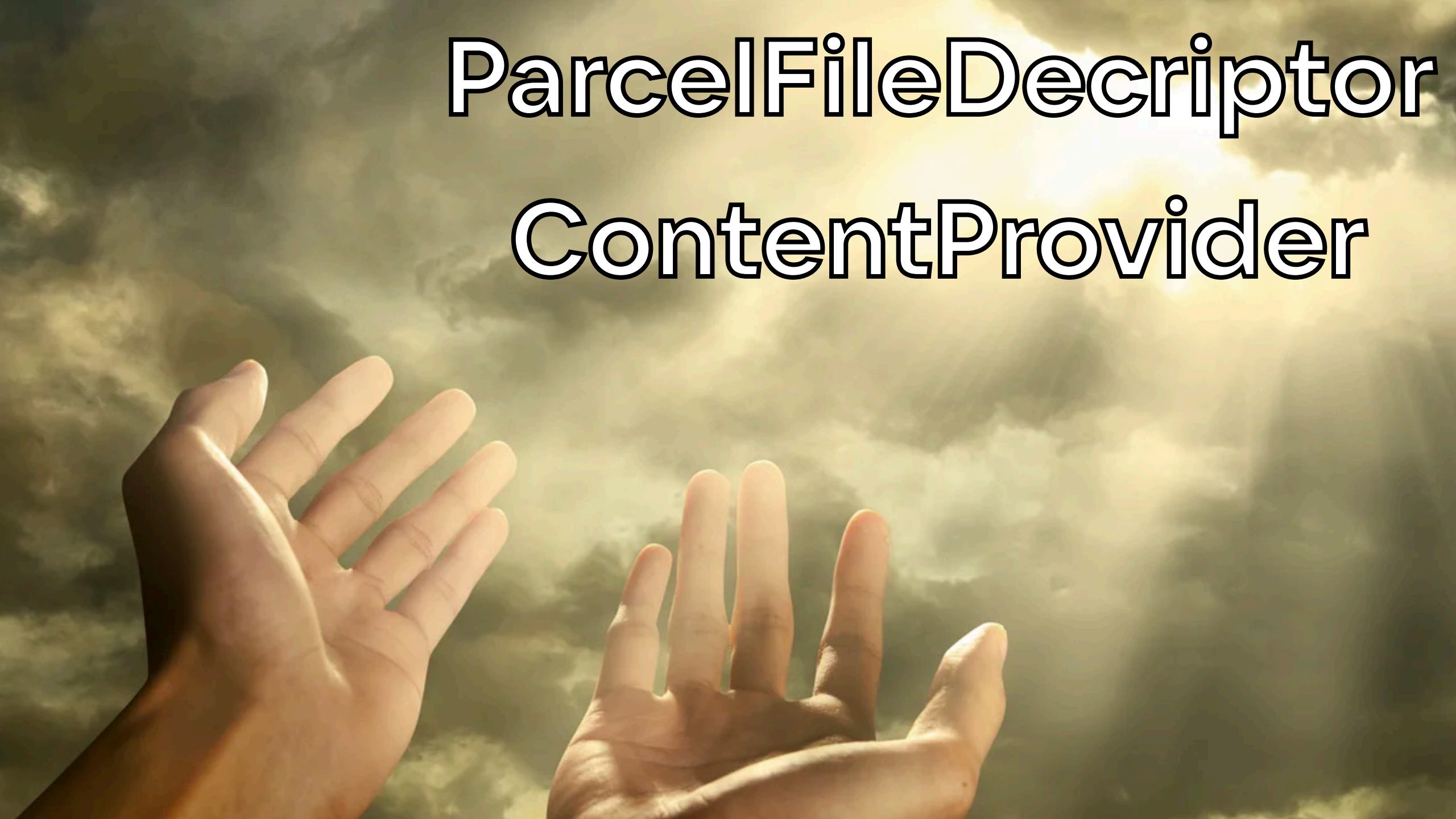


~~Downloads~~

CacheDir

Шаринг




ParcelFileDecriptor
ContentProvider

ContentProvider

```
override fun openFile(
    uri: Uri,
    mode: String
): ParcelFileDescriptor? {
    if (!isValid(uri)) return null
    return runCatching {
        val fileName = extractFileName(uri)
        createFileDescriptor(fileName)
    }.getOrNull()
}
```

ContentProvider

```
override fun openFile(
    uri: Uri,
    mode: String
): ParcelFileDescriptor? {
    if (!isValid(uri)) return null
    return runCatching {
        val fileName = extractFileName(uri)
        createFileDescriptor(fileName)
    }.getOrNull()
}
```

ContentProvider

```
override fun openFile(
    uri: Uri,
    mode: String
): ParcelFileDescriptor? {
    if (!isValid(uri)) return null
    return runCatching {
        val fileName = extractFileName(uri)
        createFileDescriptor(fileName)
    }.getOrNull()
}
```

ParcelFileDescriptor

```
fun createFileDescriptor(name: String): ParcelFileDescriptor? {
    val file = File(context.cacheDir, name)
    val sm = context.getSystemService(StorageManager::class.java)
    val thread = HandlerThread("FileSharing")
    thread.start()
    val callback = createCallback(file, thread)
    return storageManager.openProxyFileDescriptor(
        ParcelFileDescriptor.MODE_READ_ONLY,
        callback,
        Handler(thread.looper)
    )
}
```

ParcelFileDescriptor

```
fun createFileDescriptor(name: String): ParcelFileDescriptor? {
    val file = File(context.cacheDir, name)
    val sm = context.getSystemService(StorageManager::class.java)
    val thread = HandlerThread("FileSharing")
    thread.start()
    val callback = createCallback(file, thread)
    return storageManager.openProxyFileDescriptor(
        ParcelFileDescriptor.MODE_READ_ONLY,
        callback,
        Handler(thread.looper)
    )
}
```

ParcelFileDescriptor

```
fun createFileDescriptor(name: String): ParcelFileDescriptor? {
    val file = File(context.cacheDir, name)
    val sm = context.getSystemService(StorageManager::class.java)
    val thread = HandlerThread("FileSharing")
    thread.start()
    val callback = createCallback(file, thread)
    return storageManager.openProxyFileDescriptor(
        ParcelFileDescriptor.MODE_READ_ONLY,
        callback,
        Handler(thread.looper)
    )
}
```

ParcelFileDescriptor

```
fun createFileDescriptor(name: String): ParcelFileDescriptor? {
    val file = File(context.cacheDir, name)
    val sm = context.getSystemService(StorageManager::class.java)
    val thread = HandlerThread("FileSharing")
    thread.start()
    val callback = createCallback(file, thread)
    return storageManager.openProxyFileDescriptor(
        ParcelFileDescriptor.MODE_READ_ONLY,
        callback,
        Handler(thread.looper)
    )
}
```

ParcelFileDescriptor

```
fun createFileDescriptor(name: String): ParcelFileDescriptor? {
    val file = File(context.cacheDir, name)
    val sm = context.getSystemService(StorageManager::class.java)
    val thread = HandlerThread("FileSharing")
    thread.start()
    val callback = createCallback(file, thread)
    return storageManager.openProxyFileDescriptor(
        ParcelFileDescriptor.MODE_READ_ONLY,
        callback,
        Handler(thread.looper)
    )
}
```

ParcelFileDescriptorCallback

```
object : ProxyFileDescriptorCallback() {
    override fun onGetSize(): Long = file.length()
    override fun onRead(offset: Long, size: Int, data: ByteArray?): Int {
        return createCipherInputStream(file, offset).use {
            var totalRead = 0
            while (totalRead <= size) {
                val after = it.read(data, totalRead, size - totalRead)
                if (after <= 0) break
                totalRead += after
            }
            totalRead
        }
    }
    override fun onRelease() {
        thread.quitSafely()
    }
}
```

ParcelFileDescriptorCallback

```
object : ProxyFileDescriptorCallback() {
    override fun onGetSize(): Long = file.length()
    override fun onRead(offset: Long, size: Int, data: ByteArray?): Int {
        return createCipherInputStream(file, offset).use {
            var totalRead = 0
            while (totalRead <= size) {
                val after = it.read(data, totalRead, size - totalRead)
                if (after <= 0) break
                totalRead += after
            }
            totalRead
        }
    }
    override fun onRelease() {
        thread.quitSafely()
    }
}
```

ParcelFileDescriptorCallback

```
object : ProxyFileDescriptorCallback() {
    override fun onGetSize(): Long = file.length()
    override fun onRead(offset: Long, size: Int, data: ByteArray?): Int {
        return createCipherInputStream(file, offset).use {
            var totalRead = 0
            while (totalRead <= size) {
                val after = it.read(data, totalRead, size - totalRead)
                if (after <= 0) break
                totalRead += after
            }
            totalRead
        }
    }
    override fun onRelease() {
        thread.quitSafely()
    }
}
```

ParcelFileDescriptorCallback

```
object : ProxyFileDescriptorCallback() {
    override fun onGetSize(): Long = file.length()
    override fun onRead(offset: Long, size: Int, data: ByteArray?): Int {
        return createCipherInputStream(file, offset).use {
            var totalRead = 0
            while (totalRead <= size) {
                val after = it.read(data, totalRead, size - totalRead)
                if (after <= 0) break
                totalRead += after
            }
            totalRead
        }
    }
    override fun onRelease() {
        thread.quitSafely()
    }
}
```

ParcelFileDescriptor

```
fun createFileDescriptor(name: String): ParcelFileDescriptor? {
    val file = File(context.cacheDir, name)
    val sm = context.getSystemService(StorageManager::class.java)
    val thread = HandlerThread("FileSharing")
    thread.start()
    val callback = createCallback(file, thread)
    return storageManager.openProxyFileDescriptor(
        ParcelFileDescriptor.MODE_READ_ONLY,
        callback,
        Handler(thread.looper)
    )
}
```

ParcelFileDescriptor

```
fun createFileDescriptor(name: String): ParcelFileDescriptor? {
    val file = File(context.cacheDir, name)
    val sm = context.getSystemService(StorageManager::class.java)
    val thread = HandlerThread("FileSharing")
    thread.start()
    val callback = createCallback(file, thread)
    return storageManager.openProxyFileDescriptor(
        ParcelFileDescriptor.MODE_READ_ONLY,
        callback,
        Handler(thread.looper)
    )
}
```

ParcelFileDescriptor до API 26

```
@RequiresApi(Build.VERSION_CODES.O)
private fun createFileDescriptor(
    fileName: String
): ParcelFileDescriptor?
```

MemoryFile позволяет «загрузить» данные в ОЗУ

MemoryFile

```
private fun createMemoryFile(file: File): MemoryFile {
    val size = file.length()
    val memoryFile = MemoryFile(file.name, size.toInt())

    val inStream = createCipherInputStream(file)
    val outStream = memoryFile.outputStream
    inStream.copyTo(outStream)
    inStream.close()
    outStream.close()

    return memoryFile
}
```

MemoryFile

```
private fun createMemoryFile(file: File): MemoryFile {
    val size = file.length()
    val memoryFile = MemoryFile(file.name, size.toInt())

    val inStream = createCipherInputStream(file)
    val outStream = memoryFile.outputStream
    inStream.copyTo(outStream)
    inStream.close()
    outStream.close()

    return memoryFile
}
```

MemoryFile

```
private fun createMemoryFile(file: File): MemoryFile {
    val size = file.length()
    val memoryFile = MemoryFile(file.name, size.toInt())

    val inStream = createCipherInputStream(file)
    val outStream = memoryFile.outputStream
    inStream.copyTo(outStream)
    inStream.close()
    outStream.close()

    return memoryFile
}
```

MemoryFile

```
@SuppressLint("DiscouragedPrivateApi")
private fun createFileDescriptor(file: MemoryFile): ParcelFileDescriptor? {
    val method = MemoryFile::class.java.getDeclaredMethod(«getFileDescriptor")
    val fileDescriptor = method.invoke(file) as FileDescriptor

    val field = fileDescriptor.javaClass.getDeclaredField(«descriptor")
    field.isAccessible = true
    val fd = field.getInt(fileDescriptor)

    return ParcelFileDescriptor.adoptFd(fd)
}
```

MemoryFile

```
@SuppressLint("DiscouragedPrivateApi")
private fun createFileDescriptor(file: MemoryFile): ParcelFileDescriptor? {
    val method = MemoryFile::class.java.getDeclaredMethod(«getFileDescriptor")
    val fileDescriptor = method.invoke(file) as FileDescriptor

    val field = fileDescriptor.javaClass.getDeclaredField(«descriptor")
    field.isAccessible = true
    val fd = field.getInt(fileDescriptor)

    return ParcelFileDescriptor.adoptFd(fd)
}
```

MemoryFile

```
@SuppressLint("DiscouragedPrivateApi")
private fun createFileDescriptor(file: MemoryFile): ParcelFileDescriptor? {
    val method = MemoryFile::class.java.getDeclaredMethod(«getFileDescriptor")
    val fileDescriptor = method.invoke(file) as FileDescriptor

    val field = fileDescriptor.javaClass.getDeclaredField(«descriptor")
    field.isAccessible = true
    val fd = field.getInt(fileDescriptor)

    return ParcelFileDescriptor.adoptFd(fd)
}
```

MemoryFile

```
@SuppressLint("DiscouragedPrivateApi")
private fun createFileDescriptor(file: MemoryFile): ParcelFileDescriptor? {
    val method = MemoryFile::class.java.getDeclaredMethod(«getFileDescriptor")
    val fileDescriptor = method.invoke(file) as FileDescriptor

    val field = fileDescriptor.javaClass.getDeclaredField(«descriptor")
    field.isAccessible = true
    val fd = field.getInt(fileDescriptor)

    return ParcelFileDescriptor.adoptFd(fd)
}
```

MemoryFile фактически не имеет лимита

Выводы

Выводы

Android шифрует данные, но нужно и самим

Выводы

Android шифрует данные, но нужно и самим

Готовое решение есть, но оно далеко от идеала

Выводы

Android шифрует данные, но нужно и самим

Готовое решение есть, но оно далеко от идеала

Нужен произвольный доступ? Выбирайте CTR

Выводы

Android шифрует данные, но нужно и самим

Готовое решение есть, но оно далеко от идеала

Нужен произвольный доступ? Выбирайте CTR

Защита должна быть многоуровневой

Вопросы?