Fingerprint

Fingerprint-ს საკმაოდ ბევრი დადებითი მხარეები აქვს:

* ეს არის სწრაფი და მოსახერხებელი გზა მომხმარებლის იდენტურობის შესამოწმებლად
* ვერანაირად ვერ დაგავიწყდება fingerprint
* აღარ გიწევთ მობილურში პაროლისა და ლოგინის შეყვანა
* შენი თითის ანაბეჭდი არის უნიკალური ასე რომ მისი გამოყენება სხვას არ შეუძლია



თქვენ უკვე Fingerprint-ის მეშვეობით აღარ გიწევთ შრომატევადი აუთენთიფიკაციის გავლა. ამ ყველაფრისთვის თქვენ მხოლოდ დაგჭირდებათ თქვენი საკუთარი თითის ანაბეჭდი. ეს კი ბევრად მარტივად გამოსაყენებია და ნაკლებ შრომას მოითხოვს.

Android 6.0 გამოშვების შემდეგ მნიშვნელოვანი ცვლილებები მოხდა. ერთ – ერთი მათგანია თითის ანაბეჭდის ავთენტიფიკაცია.

როგორ შევქმნათ fingerprint ჩვენს საკუთარ აპლიკაციაში?

იმის შემდეგ რაც ახალ პროექტს შევქმით, ჩვენ აუცილებლად უნდა დავამატოთ ნებართვა Fingerprint-ზე

Androidmanifest***.***Xml ფაილში. ამისთვის კი ჩვენ ვწერთ :

<uses-permission android:name="android.permission.USE\_FINGERPRINT"/>

სანამ უშუალოდ კოდირებაზე გადავალთ აუცილებელია ***activity\_main.xml-***ში დავამატოთ imageview da textview საკუთარი გემოვნებით.

<?xml version="1.0" encoding="utf-8"?>

<androidx.constraintlayout.widget.ConstraintLayout

xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:tools="http://schemas.android.com/tools"

xmlns:app="http://schemas.android.com/apk/res-auto"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

tools:context=".MainActivity">

<ImageView

android:layout\_width="160dp"

android:layout\_height="160dp"

app:layout\_constraintEnd\_toEndOf="parent"

app:layout\_constraintStart\_toStartOf="parent"

android:layout\_marginStart="8dp"

app:layout\_constraintBottom\_toBottomOf="parent"

android:layout\_marginTop="8dp"

app:layout\_constraintTop\_toTopOf="parent"

android:layout\_marginBottom="8dp"

android:src="@drawable/ic\_fingerprint"

android:layout\_marginEnd="8dp"

android:id="@+id/fingerprint\_iv"/>

<TextView

android:id="@+id/fingerprint\_tv"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_marginStart="8dp"

app:layout\_constraintEnd\_toEndOf="parent"

app:layout\_constraintStart\_toStartOf="parent"

android:layout\_marginTop="32dp"

app:layout\_constraintTop\_toBottomOf="@+id/fingerprint\_iv"

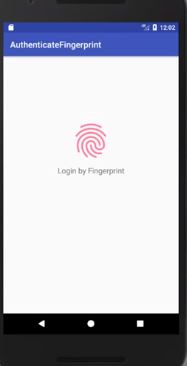
android:layout\_marginEnd="8dp"

android:text="Touch the Fingerpeint Sensor"

android:textSize="24sp"/>

</androidx.constraintlayout.widget.ConstraintLayout>

🡨 ზემოთ მოცემული კოდი დაახლოებით გამოიყურება ასე.



თითის ანაბეჭდის ავთენთიფიკაცია იყენებს KeyguardManager და FingerprintManager-ებს. ასე რომ,

onCreate () ფუნქციაში, თქვენ უნდა მოიპოვოთ ეს სერვისები:

**class** **MainActivity** : **AppCompatActivity**() {

**private** lateinit var fingerprintManager: FingerprintManager

**private** lateinit var keyguardManager: KeyguardManager

override fun **onCreate**(savedInstanceState: Bundle) {

**super**.onCreate(savedInstanceState)

setContentView(R.layout.activity\_main)

**if** (checkLockScreen()) {

}

}

**private** fun **checkLockScreen**(): Boolean {

keyguardManager = getSystemService(Context.KEYGUARD\_SERVICE)

as KeyguardManager

fingerprintManager = getSystemService(Context.FINGERPRINT\_SERVICE)

as FingerprintManager

}

}

ამის შემდეგ ჩვენ უნდა შევამოწმოთ არის თუ არა დევაისი დაპინული ან პაროლით დაცული. თუ ეს ამგვარად იქნება ჩვენ ისიც უნდა გადავამოწმოთ უკავშირდება თუ არა მოწყობილობას თითის ანაბეჭდი. ამრიგად, ამ შემოწმებას ჩვენ შევასრულებთ MainActivity.kt ფაილის checkLockScreen () მეთოდით.

**private** fun **checkLockScreen**(): Boolean {

keyguardManager = getSystemService(Context.KEYGUARD\_SERVICE)

as KeyguardManager

fingerprintManager = getSystemService(Context.FINGERPRINT\_SERVICE)

as FingerprintManager

**if** (keyguardManager.isKeyguardSecure == **false**) {

Toast.makeText(**this**,

"დახურული ეკრანის დაცვა არ არის ჩართული",

Toast.LENGTH\_LONG).show()

**return** **false**

}

**if** (ActivityCompat.checkSelfPermission(**this**,

Manifest.permission.USE\_FINGERPRINT) !=

PackageManager.PERMISSION\_GRANTED) {

Toast.makeText(**this**,

"თქვენ არ გაქვთ ნებართვა",

Toast.LENGTH\_LONG).show()

**return** **false**

}

**if** (fingerprintManager.hasEnrolledFingerprints() == **false**) {

Toast.makeText(**this**,

"თქვენ ვერ დარეგისტრირდით თითის ანაბეჭდით, გთხოვთ დარეგისტრირდეთ",

Toast.LENGTH\_LONG).show()

**return** **false**

}

**return** **true**

}

ახლა, ჩვენ უნდა შევქმნათ დაშიფვრის გასაღები, რომელიც ინახება Android Keystore სისტემაში. ასე რომ, ჩვენ უნდა მოვიპოვოთ Keystore– ზე წვდომა, შემდეგ კი გენერირების () მეთოდის დახმარებით შევქმნათ დაშიფვრის კოდი.

**class** **MainActivity** : **AppCompatActivity**() {

...

**private** lateinit var keyStore: KeyStore

**private** lateinit var keyGenerator: KeyGenerator

**private** val KEY\_NAME = "my\_key"

...

override fun **onCreate**(savedInstanceState: Bundle) {

**super**.onCreate(savedInstanceState)

setContentView(R.layout.activity\_main)

**if** (checkLockScreen()) {

generateKey()

}

}

**private** fun **checkLockScreen**(): Boolean {

}

**private** fun **generateKey**() {

**try** {

keyStore = KeyStore.getInstance("AndroidKeyStore")

} **catch** (e: Exception) {

e.printStackTrace()

}

**try** {

keyGenerator = KeyGenerator.getInstance(

KeyProperties.KEY\_ALGORITHM\_AES,

"AndroidKeyStore")

} **catch** (e: NoSuchAlgorithmException) {

**throw** RuntimeException(

"Failed to get KeyGenerator instance", e)

} **catch** (e: NoSuchProviderException) {

**throw** RuntimeException("Failed to get KeyGenerator instance", e)

}

**try** {

keyStore.load(**null**)

keyGenerator.init(

KeyGenParameterSpec.Builder(KEY\_NAME,

KeyProperties.PURPOSE\_ENCRYPT or KeyProperties.PURPOSE\_DECRYPT)

.setBlockModes(KeyProperties.BLOCK\_MODE\_CBC)

.setUserAuthenticationRequired(**true**)

.setEncryptionPaddings(

KeyProperties.ENCRYPTION\_PADDING\_PKCS7)

.build())

keyGenerator.generateKey()

} **catch** (e: NoSuchAlgorithmException) {

**throw** RuntimeException(e)

} **catch** (e: InvalidAlgorithmParameterException) {

**throw** RuntimeException(e)

} **catch** (e: CertificateException) {

**throw** RuntimeException(e)

} **catch** (e: IOException) {

**throw** RuntimeException(e)

}

}

}

როცაარუნდა გვჭირდებოდეს თითის ანაბეჭდით აუთენთიფიკაცია, ამ ყველაფერზე მოქმედებს ერთ-ერთი მეთოდი რომელსაც FingerprintManager-ს უწოდებენ. ეს მეთოდია პასუხისმგებელი ყოველი წარმატებული და წარუმატებელი აუთენთიფიკაციის.

ჩვენს პროექტში აუცილებლად უნდა დავამატოთ ერთი კლასი სახელად ***FingerprintHelper.kt***

**class** **FingerprintHelper**(**private** **val** **appContext**: **Context**) : **FingerprintManager**.**AuthenticationCallback**() {

lateinit var cancellationSignal: CancellationSignal

fun **startAuth**(manager: FingerprintManager,

cryptoObject: FingerprintManager.CryptoObject) {

cancellationSignal = CancellationSignal()

**if** (ActivityCompat.checkSelfPermission(appContext,

Manifest.permission.USE\_FINGERPRINT) !=

PackageManager.PERMISSION\_GRANTED) {

**return**

}

manager.authenticate(cryptoObject, cancellationSignal, 0, **this**, **null**)

}

override fun **onAuthenticationError**(errMsgId: Int,

errString: CharSequence) {

Toast.makeText(appContext,

"Authentication error\n" + errString,

Toast.LENGTH\_LONG).show()

}

override fun **onAuthenticationHelp**(helpMsgId: Int,

helpString: CharSequence) {

Toast.makeText(appContext,

"Authentication help\n" + helpString,

Toast.LENGTH\_LONG).show()

}

override fun **onAuthenticationFailed**() {

Toast.makeText(appContext,

"Authentication failed.",

Toast.LENGTH\_LONG).show()

}

override fun **onAuthenticationSucceeded**(

result: FingerprintManager.AuthenticationResult) {

Toast.makeText(appContext,

"Authentication succeeded.",

Toast.LENGTH\_LONG).show()

}

}