FIDELITY

To verify that the examples we found can be indeed written by developers, and is not due to reverse engineering the APK files using jadx, we developed minimal working examples using Android studio, and Proguard/R8. The APK files generated are decompiled using jadx and the fidelity of the code written by developers vs the code generated by the decompiler is ascertained.

We focused on key examples such as StringBuilder, StringBuffer, Ternary expression, method call, concatenation, Base64 use, Enum type, Native method call, and Static field constant.

Using Proguard/R8 settings such as ***isMinifyEnabled = true/false***, ***-dontoptimize***, ***-keep*** (certain classes), code compiled to APK can behave differently. For example, if we decompile APKs generated from code written with StringBuilder & StringBuffer APIs and built with ***isMinifyEnabled*** set to ***false.*** In that case, the code retrieved from the decompiler will collapse into the full string rather than individual fragments of the string. On the other hand, if ***isMinifyEnabled*** is set to ***true*** with ***-dontoptimize*** used in Proguard settings, the code as seen from the decompiler is identical to the code written by the developers.

With Static Field Constants and the use of the " + " operator for concatenation, the decompiled code shows that the individual fragments of strings are retrieved from identifiers and combined regardless of if ***isMinifyEnabled*** is set to ***true/false.*** However, if the values are not hardcoded or are retrieved dynamically, the code written with the " + " operator is identical to the code retrieved after decompilation. If String.format is combined with static field constant, the code written is similar to the code retrieved after decompilation.

Other findings including the use of ternary expression involve the code being expanded to an if statement if used with ***isminifyEnabled*** set to ***true*** and ***-dontoptimize*** and ***isminifyEnabled*** set to ***false*** the code written is identical to the code retrieved from jadx. Method calls, Native method calls, and Base64 produce virtually identical code regardless of the configurations used while string.format produces identical code only if ***isminifyEnabled*** is set to ***false*** otherwise, the code is partially identical.

Additionally, our preliminary analysis with open source code on GitHub also found examples where developers used StringBuillder while instantiating the Cipher.getInstance object. The links are provided below.

Link 1 - Use of StringBuilder for Cipher.getInstance on GitHub
Link 2 - Use of StringBuilder for Cipher.getInstance on GitHub

	Settings	**Original Code in Android Studio**	**Decompiled Code**

Native Code	minifyEnabled false	String native1 = helloWorld();
		cipher = Cipher.getInstance(native1);
			Cipher.getInstance(helloWorld());

	minifyEnabled true
	-dontoptimize
		String native1 = helloWorld();
		cipher = Cipher.getInstance(native1);
			String helloWorld = helloWorld();
			Cipher.getInstance(helloWorld);

Method Call	minifyEnabled true -dontoptimize OR minifyEnabled false	Cipher.getInstance(m12952a("AES/CBC/PKCS5Padding"));	Cipher.getInstance(m12952a("AES/CBC/PKCS5Padding"));

StringBuilder	minifyEnabled true
	-dontoptimize
	
	Shrinks code otherwise
		StringBuilder sb = **new** StringBuilder();
		sb.append("AES");
		sb.append("/EC");
		sb.append("B/PKCS7P");
		sb.append("adding");
		Cipher.getInstance(sb.toString(), provider);
			StringBuilder sb = **new** StringBuilder();
			sb.append("AES");
			sb.append("/EC");
			sb.append("B/PKCS7P");
			sb.append("adding");
			Cipher.getInstance(sb.toString(), provider);

StringBuffer	minifyEnabled true
	-dontoptimize
	
	Shrinks code otherwise
		StringBuffer stringBuffer = **new** StringBuffer();
		stringBuffer.append("DESede/CBC/");
		stringBuffer.append("NoPadding");
		this.test1 = Cipher.getInstance(stringBuffer.toString());
			StringBuffer stringBuffer = **new** StringBuffer();
			stringBuffer.append("DESede/CBC/");
			stringBuffer.append("NoPadding");
			this.f4787a =
			Cipher.getInstance(stringBuffer.toString());

String Operation (format)	minifyEnabled true -dontoptimize	Cipher.getInstance(String.format("%s/%s/%s", TypeEncryp,Type,sPadding,));	Cipher.getInstance(String.format("%s/%s/%s", "AES", "/ECB", "/PKCS7Padding"));

Concatenation	minifyEnabled true
	-dontoptimize
		this.test2 = Cipher.getInstance("DES/CBC/" + padding);
		===
		
		Cipher.getInstance(TYPE_AES + BLOCKING_MODE + PADDING_TYPE);
			this.f4788b = Cipher.getInstance("DES/CBC/" + valueOf);
			=====================================
			======
			
			Cipher.getInstance("AES/ECB/PKCS5Padding")
			;

Base64	minifyEnabled true/false -dontoptimize	Cipher.getInstance(**new** String(Base64.decode("REVTL0NCQy9QS0NTNVBhZGRpbmc=", 2)));	Cipher.getInstance(**new** String(Base64.decode("REVTL0NCQy9QS0NTNVBhZGRpbmc=", 2)));
Ternary Operator	minifyEnabled false Expands code otherwise	**this**.test3 = Cipher.getInstance(isEvenMinute() ? TRANSFORMATION_NOPANDING : TRANSFORMATION_PANDING);	**this**.test3 = Cipher.getInstance(isEvenMinute() ? TRANSFORMATION_NOPANDING : TRANSFORMATION_PANDING);

	minifyEnabled true
	-dontoptimize
		this.test3 = Cipher.getInstance(isEvenMinute() ? TRANSFORMATION_NOPANDING : TRANSFORMATION_PANDING);	if (d()) {
			 str5 = "AES/ECB/NoPadding";
			} else {
			  str5 = "DES/CBC/PKCS5Padding";
			  }
			this.f4819c = Cipher.getInstance(str5);

Static Field Constant	Shrinks code if used without String.format in all cases. minifyEnabled false (produce identical code with minify set to false)	Cipher cipher2 = Cipher.getInstance(String.format("%1$s/%2$s/%3$s", Constants.TYPE_AES, Constants.BLOCKING_MODE, Constants.PADDING_TYPE));	Cipher cipher2 = Cipher.getInstance(String.format("%1$s/%2$s/%3$s", Constants.TYPE_AES, Constants.BLOCKING_MODE, Constants.PADDING_TYPE));