Encrypting strings in Android: Let's make better mistakes

1 DEC 2014

- FACEBOOK
- TWITTER
- PINTEREST
- GOOGLE+

ENCRYPTING STRINGS IN ANDROID: LET'S MAKE BETTER MISTAKES

Posted On December 1, 2014 By Isaac Potoczny-Jones And has No Comment

Update: Here's the video of Isaac's talk on this topic and the Github repo for the AES library.

Follow Us

For writing about computer security, authentication, and Tozny news, subscribe to our RSS feed or

Follow @tozny

Recent Posts

FedScoop: NIST IoT project explores how to ditch passwords, maintain privacy
November 16, 2015 - 8:05 am

Portland Business Journal covers Tozny's NSTIC project
November 9, 2015 - 2:17 pm

How the Federal government is attempting to protect the Internet

If you do a web search for “encrypting Strings in Android”, you’ll find a lot of example code, and they all look pretty similar. They definitely input a String and output gibberish that looks like encrypted text, but they are often incorrect. Crypto is tricky: it’s hard to tell that the gibberish that’s being printed is not good crypto, and it’s hard to tell that the code example you picked up from Stack Overflow has serious flaws.

The problem here is that sites like Google and Stack Overflow rank results based on popularity, but the correctness of crypto isn’t something we can vote about. It’s not a popularity contest. To use it correctly, you have to understand the properties of the algorithm and the security goals of your code. Maybe the bad crypto someone pasted up on the Internet was acceptable for their needs, but there’s a good chance it’s completely unacceptable for yours.

We want to help make things better, but let’s start by pointing out specifically why this is so problematic. Google for “encrypting strings in Android” and the first hit is a great example of how to do it all wrong. By virtue of being the first hit in Google, this code has propagated all over the place, including hundreds of Github projects.

Here are the typical mistakes you’ll see:

- **Bad key generation**: Ideally, AES keys should be securely generated and random. Alternately, depending on the use case,

attempting to protect the Internet of Things
November 3, 2015 - 9:49 am

IoT security & privacy requires overcoming a legacy of insecurity
November 2, 2015 - 11:42 am

Archives

November 2015

October 2015

September 2015

June 2015

May 2015

April 2015

March 2015

January 2015

December 2014

November 2014

October 2014

September 2014

August 2014

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

you can use a standard password-based method to generate a key that's as strong as the password. Lots of sites recommend using the bytes from a password as the seed to the random number generator. That's not right.

- **Out of date key generation**: In 2013, folks realized that the SecureRandom implementation in Android is flawed and developers need to explicitly initialize the PRNG with entropy. This error caused the loss of some bitcoin.
- **Use of ECB**: The default mode of AES in Android is ECB, which encrypts each block the same, and so is subject to analysis and replay attacks. Here's an example of why ECB isn't so great. Or imagine you're encrypting a bunch of passwords: you couldn't necessarily decrypt the passwords without the key, but you **could** tell which sites use the same passwords. A better mode is CBC, and that mode has been available for quite some time in Android.
- **Bad Padding**: When you're encrypting data that's less than the size of an AES block, you need to specify a secure padding scheme. Even then, it's kinda tricky.
- **No integrity**: Lots of people think AES has integrity checking built in. The thinking goes, "if it decrypts correctly, it was generated by the person with the private key". Actually, AES CBC allows an attacker to modify the messages. In our vanilla searches, we didn't find a single example of people doing

@Tozny on Twitter

RT @Nextgov: NIST Testing out Passwordless Smart Home https://t.co/5fXkGCsu3e https://t.co/C8ba71bj3M

NIST #IoT project explores how to ditch passwords, maintain privacy by @fedscoop @NSTICNPO @globesherpa @iotashome https://t.co/W814tUEWVm

RT @InfosecurityMag: NIST Awards $1.86Mn IoT Privacy Grant: The pilot will focus on allowing consumers to securely store and share ... http⊠

RT @SRI_Intl: SRI is part of #NSTIC privacy & security pilot program team with @IOTAShome @galois

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

integrity checking with AES CBC.

- **Incorrect IV**: When using e.g. CBC mode, a random IV is required. This makes dealing with the ciphertext a little more complicated since you need to keep the (non-secret) IV around to decrypt it. Since it's tricky, sometimes people use a fixed IV, which is often bad. Android does some pretty weird things if you don't specify an IV (and I've heard that some Android versions don't even let you specify an IV):
  - In KitKat, Android generates a random looking IV during encrypt, which was different (at least for each new instance of Cipher) and used all zeros during decrypt; between runs, the IV varies, so it will never decrypt the same way twice. This is arguably the "right" behavior if someone doesn't set the IV, but if you don't know what you're doing, you'll encrypt something and never be able to decrypt it.
  - On 4.3, the app used an IV of all zeros for both encrypt and decrypt so it always encrypts and decrypts the same.
  - On my rooted CyanogenMod 4.2.2 device, it used a random looking IV that was the same between runs and between encrypt and decrypt. I don't know where it's getting / storing this IV.
- **Weak algorithms**: Use of DES or MD5 is problematic since both of those algorithms have been demonstrated to be weak and should no longer be used.

@IOTAShome, @galois, @globesherpa, @tozny https://t.co/ztXf

Portland Business Journal discusses Tozny's new #IoT and #transit security and privacy project. @NSTICNPO https://t.co/x6zF8oVhdE

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

In fact, misuse of cryptographic libraries is a common source of bugs as this paper on 269 cryptography-related CVEs demonstrates. Check out the appendix for a plethora of bad Android crypto examples.

A Java class that can help

Download on Github

Although complete crypto libraries like Keyczar are widely available, a lot of developers are looking for a short class they can paste into their code. As much as crypto professionals don't like this behavior since it's likely to result in bad code, it's the reality.

To fix this, we provide a Github repo that's a short standalone Java class to correctly use the built-in AES libraries. This code can be added to an Android project. We're sure that it needs work, but that it's far better than anything currently coming up in Google results. We would very much appreciate review and pull requests if you can improve this class.

Here are the properties of this class. We believe that these properties are consistent with what a lot of people are looking for:

- **Paste-ability:** A very simple Java class that works across most or all versions of Android. The class should be easy to paste into an existing codebase.
- **Works for strings:** It should encrypt arbitrary strings or byte

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

arrays. This means it needs to effectively handle multiple blocks (CBC) and partial blocks (padding). It consistently serializes and deserializes ciphertext, IVs, and key material using base64 to make it easy to store.

- **Algorithm & Mode**: We chose: AES 128, CBC, and PKCS5 padding. We use a 128 bit key size for its widespread support and because it’s not clear that 256 is stronger. We would have picked GCM for its built-in integrity checking, but that’s only available since Android Jelly Bean, which leaves out about 1/4 of active Android devices
- **IV Handling:** We securely generate a random IV before each encryption and provide a simple class to keep the IV and ciphertext together so they’re easy to keep track of and store. We set the IV and then request it back from the Cipher class for compatibility across various Android versions.
- **Key generation**: Random key generation with the updated generation code recommended for Android. If you want password-based keys, we provide functions to salt and generate them.
- **Integrity**: We’ve also added more-or-less transparent integrity checking in the form of a SHA 256 MAC with a constant-time equality check. This is in the form of a “combined key” where 128 bits of the key are used for encryption and 256 bits are used for integrity, then the keys are kept together.

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

I'm sure it's not perfect, but maybe over time we can get the bad code taken down and replaced with code that's at least not completely wrong.

Thanks!

Much credit to Chris Swenson and Thomas DuBuisson for advice and code to make this library better.

Appendix: Let's look for crypto

A lot of programmers need to depend on getting good search results for building code that's outside of their areas of expertise. Usually, you can tell that such code is working or not working, but with crypto, not so much. In summary, the top Google results are all either wrong or out of date. Out of the 10 top search results we found (5 each for different queries) only **one** of them correctly generates keys and IVs, and **none** of them have the up to date key generation from 2013 and none of them use any integrity checking.

Let's look at the results from a few specific searches:

Search for: Encrypting strings in Android

Googling for "encrypting strings in Android" here are the top 5 results:

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

Result	Problems	Notes
Bad crypto 1	Bad key generation, use of ECB, bad padding. Also doesn't specify the byte encoding.	Users are complaining about padding exceptions, which isn't what you want. This code seems to be the basis of a lot of bad crypto on the Internet. Anyone who has copied this code has a lot of problems. Someone needs to get them to take it down.
Bad crypto 2	Weak algorithm: MD5	The person asking the question doesn't know if they want encryption or hashing. The people give them hashing.
Bad	Weak	Do I remember

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

crypto 3	algorithm: DES	correctly that in ancient history, Android didn’t have AES?
Bad crypto 4	Bad key generation, use of ECB. Also doesn’t specify the byte encoding.	This seems like a rehash of example 1, but the code looks like it’s on graph paper, which makes it seem more secure.
Pretty good crypto	Out of date key generation	This is a good article from 2012 about why all those above examples of “bad key generation” are wrong. It’s just out of date now.

Search for: Android AES CBC example

So let’s say you read up a bit and know you should use AES and

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

CBC (instead of DES and MD5). Here are the top results from Googling “Android AES cbc example”:

Result	Problem	Notes
Bad crypto 5	Bad key generation, use of ECB, bad padding.	I’m starting to notice that this hero is trying to warn everyone about the bad crypto.
Bad crypto 6	Bad key generation, use of ECB, bad padding.	The question itself sets the stage with terrible code. A fixed string converted to bytes as the key, a static IV (which isn’t actually used), and it looks like it sometimes uses ECB and sometimes uses CBC. The top reply uses

		a fixed IV (as OP requested) although it’s a different one from OP.
Bad crypto 7	Bad key generation, bad IV	This post looks pretty promising because it talks about different cryptographic providers, and why AES is nice. It accepts an IV, but if you don’t pass it in, it uses a fixed IV instead of a random one.
Bad crypto 8	Bad key generation, use of ECB, bad padding.	This is the same as result 1
Bad	Out of date	Nice that it actually

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

crypto 9	key generation, bad IV (zeros)	uses CBC.

PREVIOUS ARTICLE
GODADDY'S SSL CERTS DON'T WORK IN JAVA - THE RIGHT SOLUTION

NEXT ARTICLE
VIDEO: COMMON CRYPTO MISTAKES IN ANDROID

@Tozny on Twitter

RT @Nextgov: NIST Testing out Passwordless Smart Home https://t.co/5fXkGCsu3e https://t.co/C8ba71bj3M

NIST #IoT project explores how to ditch passwords, maintain privacy by

Recent Posts

FedScoop: NIST IoT project explores how to ditch passwords, maintain privacy

Portland Business Journal covers Tozny's NSTIC project

How the Federal government is attempting to protect the Internet of

Misc.

Follow @tozny

Subscribe to RSS feed

Photo and other credits

Galois, Inc.

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

@fedscoop @NSTICNPO @globesherpa @iotashome https://t.co/W814tUEWVm

RT @InfosecurityMag: NIST Awards $1.86Mn IoT Privacy Grant: The pilot will focus on allowing consumers to securely store and share ... http

Things

IoT security & privacy requires overcoming a legacy of insecurity

Podcast: Tonzy CEO Interviewed by Regarding ID

© Copyright Tozny, LLC. a Galois company.

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com